Vector2i **local_to_map**(local_position: Vector2) const

Returns the map coordinates of the cell containing the given `local_position`. If `local_position` is in global coordinates, consider using Node2D.to_local before passing it to this method. See also map_to_local.

Vector2i **map_pattern**(position_in_tilemap: Vector2i, coords_in_pattern: Vector2i, pattern: TileMapPattern)

Returns for the given coordinates `coords_in_pattern` in a TileMapPattern the corresponding cell coordinates if the pattern was pasted at the `position_in_tilemap` coordinates (see set_pattern). This mapping is required as in half-offset tile shapes, the mapping might not work by calculating `position_in_tile_map + coords_in_pattern`.

Vector2 **map_to_local**(map_position: Vector2i) const

Returns the centered position of a cell in the **TileMapLayer**'s local coordinate space. To convert the returned value into global coordinates, use Node2D.to_global. See also local_to_map.

Note: This may not correspond to the visual position of the tile, i.e. it ignores the TileData.texture_origin property of individual tiles.

---

void **notify_runtime_tile_data_update**()

Notifies the **TileMapLayer** node that calls to _use_tile_data_runtime_update or _tile_data_runtime_update will lead to different results. This will thus trigger a **TileMapLayer** update.

Warning: Updating the **TileMapLayer** is computationally expensive and may impact performance. Try to limit the number of calls to this function to avoid unnecessary update.

Note: This does not trigger a direct update of the **TileMapLayer**, the update will be done at the end of the frame as usual (unless you call update_internals).

---

void **set_cell**(coords: Vector2i, source_id: int = -1, atlas_coords: Vector2i = Vector2i(-1, -1), alternative_tile: int = 0) 🔗

Sets the tile identifiers for the cell at coordinates `coords`. Each tile of the TileSet is identified using three parts:

- The source identifier `source_id` identifies a TileSetSource identifier. See TileSet.set_source_id,
- The atlas coordinate identifier `atlas_coords` identifies a tile coordinates in the atlas (if the source is a TileSetAtlasSource). For TileSetScenesCollectionSource it should always be `Vector2i(0, 0)`,
- The alternative tile identifier `alternative_tile` identifies a tile alternative in the atlas (if the source is a TileSetAtlasSource), and the scene for a TileSetScenesCollectionSource.

If `source_id` is set to `-1`, `atlas_coords` to `Vector2i(-1, -1)`, or `alternative_tile` to `-1`, the cell will be erased. An erased cell gets **all** its identifiers automatically set to their respective invalid values, namely `-1`, `Vector2i(-1, -1)` and `-1`.

---

void **set_cells_terrain_connect**(cells: Array[Vector2i], terrain_set: int, terrain: int, ignore_empty_terrains: bool = true) 🔗

Update all the cells in the `cells` coordinates array so that they use the given `terrain` for the given `terrain_set`. If an updated cell has the same terrain as one of its neighboring cells, this function tries to join the two. This function might update neighboring tiles if needed to create correct terrain transitions.

If `ignore_empty_terrains` is true, empty terrains will be ignored when trying to find the best fitting tile for the given terrain constraints.

Note: To work correctly, this method requires the **TileMapLayer**'s TileSet to have terrains set up with all required terrain combinations. Otherwise, it may produce unexpected results.

void **set_cells_terrain_path**(path: Array[Vector2i], terrain_set: int, terrain: int, ignore_empty_terrains: bool = true) 🔗

Update all the cells in the `path` coordinates array so that they use the given `terrain` for the given `terrain_set`. The function will also connect two successive cell in the path with the same terrain. This function might update neighboring tiles if needed to create correct terrain transitions.

If `ignore_empty_terrains` is true, empty terrains will be ignored when trying to find the best fitting tile for the given terrain constraints.

Note: To work correctly, this method requires the **TileMapLayer**'s TileSet to have terrains set up with all required terrain combinations. Otherwise, it may produce unexpected results.

void **set_navigation_map**(map: RID) 🔗

Sets a custom `map` as a NavigationServer2D navigation map. If not set, uses the default World2D navigation map instead.

void **set_pattern**(position: Vector2i, pattern: TileMapPattern) 🔗

Pastes the TileMapPattern at the given `position` in the tile map. See also get_pattern.

---

void **update_internals**() 🔗

Triggers a direct update of the **TileMapLayer**. Usually, calling this function is not needed, as **TileMapLayer** node updates automatically when one of its properties or cells is modified.

However, for performance reasons, those updates are batched and delayed to the end of the frame. Calling this function will force the **TileMapLayer** to update right away instead.

Warning: Updating the **TileMapLayer** is computationally expensive and may impact performance. Try to limit the number of updates and how many tiles they impact.

Timer

Inherits: Node **<** Object

A countdown timer.

Description

The **Timer** node is a countdown timer and is the simplest way to handle time-based logic in the engine. When a timer reaches the end of its wait_time, it will emit the timeout signal.

After a timer enters the tree, it can be manually started with start. A timer node is also started automatically if autostart is `true`.

Without requiring much code, a timer node can be added and configured in the editor. The timeout signal it emits can also be connected through the Node dock in the editor:

```
func _on_timer_timeout():
    print("Time to attack!")
```

Note: To create a one-shot timer without instantiating a node, use SceneTree.create_timer.

Note: Timers are affected by Engine.time_scale. The higher the time scale, the sooner timers will end. How often a timer processes may depend on the framerate or Engine.physics_ticks_per_second.

Tutorials

- 2D Dodge The Creeps Demo [https://godotengine.org/asset-library/asset/2712]

Properties

bool	autostart	`false`
bool	one_shot	`false`
bool	paused	
TimerProcessCallback	process_callback	`1`
float	time_left	
float	wait_time	`1.0`

Methods

bool	is_stopped() const
void	start(time_sec: float = -1)
void	stop()

Signals

timeout() 🔗

Emitted when the timer reaches the end.

Enumerations

enum **TimerProcessCallback**: 🔗

TimerProcessCallback **TIMER_PROCESS_PHYSICS** = `0`

Update the timer every physics process frame (see Node.NOTIFICATION_INTERNAL_PHYSICS_PROCESS).

TimerProcessCallback **TIMER_PROCESS_IDLE** = `1`

Update the timer every process (rendered) frame (see Node.NOTIFICATION_INTERNAL_PROCESS).

Property Descriptions

bool **autostart** = `false` 🔗

- void **set_autostart**(value: bool)
- bool **has_autostart**()

If `true`, the timer will start immediately when it enters the scene tree.

Note: After the timer enters the tree, this property is automatically set to `false`.

---

bool **one_shot** = `false` 🔗

- void **set_one_shot**(value: bool)
- bool **is_one_shot**()

If `true`, the timer will stop after reaching the end. Otherwise, as by default, the timer will automatically restart.

---

bool **paused**

- void **set_paused**(value: bool)
- bool **is_paused**()

If `true`, the timer is paused. A paused timer does not process until this property is set back to `false`, even when start is called.

TimerProcessCallback **process_callback** = `1`

- void **set_timer_process_callback**(value: TimerProcessCallback)
- TimerProcessCallback **get_timer_process_callback**()

Specifies when the timer is updated during the main loop (see TimerProcessCallback).

---

float **time_left**

- float **get_time_left**()

The timer's remaining time in seconds. This is always `0` if the timer is stopped.

Note: This property is read-only and cannot be modified. It is based on wait_time.

---

float **wait_time** = `1.0` 🔗

- void **set_wait_time**(value: float)
- float **get_wait_time**()

The time required for the timer to end, in seconds. This property can also be set every time start is called.

Note: Timers can only process once per physics or process frame (depending on the process_callback). An unstable framerate may cause the timer to end inconsistently, which is especially noticeable if the wait time is lower than roughly `0.05` seconds. For very short timers, it is recommended to write your own code instead of using a **Timer** node. Timers are also affected by Engine.time_scale.

Method Descriptions

bool **is_stopped**() const 🔗

Returns `true` if the timer is stopped or has not started.

---

void **start**(time_sec: float = -1) 🔗

Starts the timer, if it was not started already. Fails if the timer is not inside the tree. If `time_sec` is greater than `0`, this value is used for the wait_time.

Note: This method does not resume a paused timer. See paused.

---

void **stop**() 🔗

Stops the timer.

TouchScreenButton

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Button for touch screen devices for gameplay use.

Description

TouchScreenButton allows you to create on-screen buttons for touch devices. It's intended for gameplay use, such as a unit you have to touch to move. Unlike Button, TouchScreenButton supports multitouch out of the box. Several TouchScreenButtons can be pressed at the same time with touch input.

This node inherits from Node2D. Unlike with Control nodes, you cannot set anchors on it. If you want to create menus or user interfaces, you may want to use Button nodes instead. To make button nodes react to touch events, you can enable the Emulate Mouse option in the Project Settings.

You can configure TouchScreenButton to be visible only on touch devices, helping you develop your game both for desktop and mobile devices.

Properties

String	action	""
BitMap	bitmask	

bool	passby_press	`false`
Shape2D	shape	
bool	shape_centered	`true`
bool	shape_visible	`true`
Texture2D	texture_normal	
Texture2D	texture_pressed	
VisibilityMode	visibility_mode	`0`

Methods

bool	is_pressed() const

Signals

pressed() 🔗

Emitted when the button is pressed (down).

---

released()

Emitted when the button is released (up).

Enumerations

enum **VisibilityMode**:

VisibilityMode **VISIBILITY_ALWAYS** = `0`

Always visible.

VisibilityMode **VISIBILITY_TOUCHSCREEN_ONLY** = `1`

Visible on touch screens only.

Property Descriptions

String **action** = `""`

- void **set_action**(value: String)
- String **get_action**()

The button's action. Actions can be handled with InputEventAction.

BitMap **bitmask**

- void **set_bitmask**(value: BitMap)
- BitMap **get_bitmask**()

The button's bitmask.

---

bool **passby_press** = `false`

- void **set_passby_press**(value: bool)
- bool **is_passby_press_enabled**()

If `true`, the pressed and released signals are emitted whenever a pressed finger goes in and out of the button, even if the pressure started outside the active area of the button.

Note: This is a "pass-by" (not "bypass") press mode.

Shape2D **shape**

- void **set_shape**(value: Shape2D)
- Shape2D **get_shape**()

The button's shape.

---

bool **shape_centered** = `true`

- void **set_shape_centered**(value: bool)
- bool **is_shape_centered**()

If `true`, the button's shape is centered in the provided texture. If no texture is used, this property has no effect.

---

bool **shape_visible** = `true`

- void **set_shape_visible**(value: bool)
- bool **is_shape_visible**()

If `true`, the button's shape is visible in the editor.

Texture2D **texture_normal** 🔗

- void **set_texture_normal**(value: Texture2D)
- Texture2D **get_texture_normal**()

The button's texture for the normal state.

Texture2D **texture_pressed** 🔗

- void **set_texture_pressed**(value: Texture2D)
- Texture2D **get_texture_pressed**()

The button's texture for the pressed state.

VisibilityMode **visibility_mode** = `0` 🔗

- void **set_visibility_mode**(value: VisibilityMode)
- VisibilityMode **get_visibility_mode**()

The button's visibility mode. See VisibilityMode for possible values.

Method Descriptions

bool **is_pressed**() const 🔗

Returns `true` if this button is currently pressed.

Tree

Inherits: Control **<** CanvasItem **<** Node **<** Object

A control used to show a set of internal TreeItems in a hierarchical structure.

Description

A control used to show a set of internal TreeItems in a hierarchical structure. The tree items can be selected, expanded and collapsed. The tree can have multiple columns with custom controls like LineEdits, buttons and popups. It can be useful for structured displays and interactions.

Trees are built via code, using TreeItem objects to create the structure. They have a single root, but multiple roots can be simulated with hide_root:

GDScript

```
func _ready():
    var tree = Tree.new()
    var root = tree.create_item()
    tree.hide_root = true
    var child1 = tree.create_item(root)
    var child2 = tree.create_item(root)
    var subchild1 = tree.create_item(child1)
    subchild1.set_text(0, "Subchild1")
```

C#

```
public override void _Ready()
{
    var tree = new Tree();
    TreeItem root = tree.CreateItem();
    tree.HideRoot = true;
    TreeItem child1 = tree.CreateItem(root);
    TreeItem child2 = tree.CreateItem(root);
    TreeItem subchild1 = tree.CreateItem(child1);
    subchild1.SetText(0, "Subchild1");
}
```

To iterate over all the TreeItem objects in a **Tree** object, use TreeItem.get_next and TreeItem.get_first_child after getting the root through get_root. You can use Object.free on a TreeItem to remove it from the **Tree**.

Incremental search: Like ItemList and PopupMenu, **Tree** supports searching within the list while the control is focused. Press a key that matches the first letter of an item's name to select the first item starting with the given letter. After that point, there are two ways to perform incremental search: 1) Press the same key again before the timeout duration to select the next item starting with the same letter. 2) Press letter keys that match the rest of the word before the timeout duration to match to select the item in question directly. Both of these actions will be reset to the beginning of the list if the timeout duration has passed since the last keystroke was registered. You can adjust the timeout duration by changing ProjectSettings.gui/timers/incremental_search_max_interval_msec.

Properties

bool	allow_reselect	`false`
bool	allow_rmb_select	`false`
bool	allow_search	`true`

bool	clip_contents	`true` (overrides Control)
bool	column_titles_visible	`false`
int	columns	`1`
int	drop_mode_flags	`0`
bool	enable_recursive_folding	`true`
FocusMode	focus_mode	`2` (overrides Control)
bool	hide_folding	`false`
bool	hide_root	`false`
bool	scroll_horizontal_enabled	`true`
bool	scroll_vertical_enabled	`true`
SelectMode	select_mode	`0`

Methods

void	clear()
TreeItem	create_item(parent: TreeItem = null, index: int = -1)
void	deselect_all()
bool	edit_selected(force_edit: bool = false)
void	ensure_cursor_is_visible()
int	get_button_id_at_position(position: Vector2) const
int	get_column_at_position(position: Vector2) const
int	get_column_expand_ratio(column: int) const
String	get_column_title(column: int) const
HorizontalAlignment	get_column_title_alignment(column: int) const
TextDirection	get_column_title_direction(column: int) const

String	get_column_title_language(column: int) const
int	get_column_width(column: int) const
Rect2	get_custom_popup_rect() const
int	get_drop_section_at_position(position: Vector2) const
TreeItem	get_edited() const
int	get_edited_column() const
Rect2	get_item_area_rect(item: TreeItem, column: int = -1, button_index: int = -1) const
TreeItem	get_item_at_position(position: Vector2) const
TreeItem	get_next_selected(from: TreeItem)
int	get_pressed_button() const
TreeItem	get_root() const
Vector2	get_scroll() const

TreeItem	get_selected() const
int	get_selected_column() const
bool	is_column_clipping_content(column: int) const
bool	is_column_expanding(column: int) const
void	scroll_to_item(item: TreeItem, center_on_item: bool = false)
void	set_column_clip_content(column: int, enable: bool)
void	set_column_custom_minimum_width(column: int, min_width: int)
void	set_column_expand(column: int, expand: bool)
void	set_column_expand_ratio(column: int, ratio: int)
void	set_column_title(column: int, title: String)

void	set_column_title_alignment(column: int, title_alignment: HorizontalAlignment)
void	set_column_title_direction(column: int, direction: TextDirection)
void	set_column_title_language(column: int, language: String)
void	set_selected(item: TreeItem, column: int)

Theme Properties

Color	children_hl_line_color	`Color(0.27, 0.27, 0.27, 1)`
Color	custom_button_font_highlight	`Color(0.95, 0.95, 0.95, 1)`
Color	drop_position_color	`Color(1, 1, 1, 1)`
Color	font_color	`Color(0.7, 0.7, 0.7, 1)`
Color	font_disabled_color	`Color(0.875, 0.875, 0.875, 0.5)`

Color	font_outline_color	`Color(0, 0, 0, 1)`
Color	font_selected_color	`Color(1, 1, 1, 1)`
Color	guide_color	`Color(0.7, 0.7, 0.7, 0.25)`
Color	parent_hl_line_color	`Color(0.27, 0.27, 0.27, 1)`
Color	relationship_line_color	`Color(0.27, 0.27, 0.27, 1)`
Color	title_button_color	`Color(0.875, 0.875, 0.875, 1)`
int	button_margin	`4`
int	children_hl_line_width	`1`
int	draw_guides	`1`
int	draw_relationship_lines	`0`
int	h_separation	`4`

int	icon_max_width	`0`
int	inner_item_margin_bottom	`0`
int	inner_item_margin_left	`0`
int	inner_item_margin_right	`0`
int	inner_item_margin_top	`0`
int	item_margin	`16`
int	outline_size	`0`
int	parent_hl_line_margin	`0`
int	parent_hl_line_width	`1`
int	relationship_line_width	`1`
int	scroll_border	`4`
int	scroll_speed	`12`
int	scrollbar_h_separation	`4`

int	scrollbar_margin_bottom	`-1`
int	scrollbar_margin_left	`-1`
int	scrollbar_margin_right	`-1`
int	scrollbar_margin_top	`-1`
int	scrollbar_v_separation	`4`
int	v_separation	`4`
Font	font	
Font	title_button_font	
int	font_size	
int	title_button_font_size	
Texture2D	arrow	
Texture2D	arrow_collapsed	

Texture2D	arrow_collapsed_mirrored
Texture2D	checked
Texture2D	checked_disabled
Texture2D	indeterminate
Texture2D	indeterminate_disabled
Texture2D	select_arrow
Texture2D	unchecked
Texture2D	unchecked_disabled
Texture2D	updown
StyleBox	button_pressed
StyleBox	cursor
StyleBox	cursor_unfocused
StyleBox	custom_button

StyleBox	custom_button_hover
StyleBox	custom_button_pressed
StyleBox	focus
StyleBox	panel
StyleBox	selected
StyleBox	selected_focus
StyleBox	title_button_hover
StyleBox	title_button_normal
StyleBox	title_button_pressed

Signals

button_clicked(item: TreeItem, column: int, id: int, mouse_button_index: int) 🔗

Emitted when a button on the tree was pressed (see TreeItem.add_button).

cell_selected() 🔗

Emitted when a cell is selected.

---

check_propagated_to_item(item: TreeItem, column: int) 🔗

Emitted when TreeItem.propagate_check is called. Connect to this signal to process the items that are affected when TreeItem.propagate_check is invoked. The order that the items affected will be processed is as follows: the item that invoked the method, children of that item, and finally parents of that item.

---

column_title_clicked(column: int, mouse_button_index: int) 🔗

Emitted when a column's title is clicked with either @GlobalScope.MOUSE_BUTTON_LEFT or @GlobalScope.MOUSE_BUTTON_RIGHT.

---

custom_item_clicked(mouse_button_index: int) 🔗

Emitted when an item with TreeItem.CELL_MODE_CUSTOM is clicked with a mouse button.

---

custom_popup_edited(arrow_clicked: bool) 🔗

Emitted when a cell with the TreeItem.CELL_MODE_CUSTOM is clicked to be edited.

---

empty_clicked(click_position: Vector2, mouse_button_index: int) 🔗

Emitted when a mouse button is clicked in the empty space of the tree.

item_activated()

Emitted when an item is double-clicked, or selected with a `ui_accept` input event (e.g. using `Enter` or `Space` on the keyboard).

item_collapsed(item: TreeItem)

Emitted when an item is collapsed by a click on the folding arrow.

item_edited()

Emitted when an item is edited.

item_icon_double_clicked()

Emitted when an item's icon is double-clicked. For a signal that emits when any part of the item is double-clicked, see item_activated.

item_mouse_selected(mouse_position: Vector2, mouse_button_index: int)

Emitted when an item is selected with a mouse button.

item_selected()

Emitted when an item is selected.

multi_selected(item: TreeItem, column: int, selected: bool) 🔗

Emitted instead of item_selected if select_mode is set to SELECT_MULTI.

---

nothing_selected() 🔗

Emitted when a left mouse button click does not select any item.

Enumerations

enum **SelectMode**: 🔗

SelectMode **SELECT_SINGLE** = `0`

Allows selection of a single cell at a time. From the perspective of items, only a single item is allowed to be selected. And there is only one column selected in the selected item.

The focus cursor is always hidden in this mode, but it is positioned at the current selection, making the currently selected item the currently focused item.

SelectMode **SELECT_ROW** = `1`

Allows selection of a single row at a time. From the perspective of items, only a single items is allowed to be selected. And all the columns are selected in the selected item.

The focus cursor is always hidden in this mode, but it is positioned at the first column of the current selection, making the currently selected item the currently focused item.

SelectMode **SELECT_MULTI** = `2`

Allows selection of multiple cells at the same time. From the perspective of items, multiple items are allowed to be selected. And there can be multiple columns selected in each selected item.

The focus cursor is visible in this mode, the item or column under the cursor is not necessarily selected.

---

enum **DropModeFlags**: 🔗

DropModeFlags **DROP_MODE_DISABLED** = `0`

Disables all drop sections, but still allows to detect the "on item" drop section by get_drop_section_at_position.

Note: This is the default flag, it has no effect when combined with other flags.

DropModeFlags **DROP_MODE_ON_ITEM** = `1`

Enables the "on item" drop section. This drop section covers the entire item.

When combined with DROP_MODE_INBETWEEN, this drop section halves the height and stays centered vertically.

DropModeFlags **DROP_MODE_INBETWEEN** = `2`

Enables "above item" and "below item" drop sections. The "above item" drop section covers the top half of the item, and the "below item" drop section covers the bottom half.

When combined with DROP_MODE_ON_ITEM, these drop sections halves the height and stays on top / bottom accordingly.

Property Descriptions

bool **allow_reselect** = `false`

- void **set_allow_reselect**(value: bool)
- bool **get_allow_reselect**()

If `true`, the currently selected cell may be selected again.

---

bool **allow_rmb_select** = `false`

- void **set_allow_rmb_select**(value: bool)
- bool **get_allow_rmb_select**()

If `true`, a right mouse button click can select items.

---

bool **allow_search** = `true`

- void **set_allow_search**(value: bool)
- bool **get_allow_search**()

If `true`, allows navigating the **Tree** with letter keys through incremental search.

---

bool **column_titles_visible** = `false`

- void **set_column_titles_visible**(value: bool)
- bool **are_column_titles_visible**()

If `true`, column titles are visible.

---

int **columns** = `1`

- void **set_columns**(value: int)
- int **get_columns**()

The number of columns.

int **drop_mode_flags** = `0`

- void **set_drop_mode_flags**(value: int)
- int **get_drop_mode_flags**()

The drop mode as an OR combination of flags. See DropModeFlags constants. Once dropping is done, reverts to DROP_MODE_DISABLED. Setting this during Control._can_drop_data is recommended.

This controls the drop sections, i.e. the decision and drawing of possible drop locations based on the mouse position.

bool **enable_recursive_folding** = `true`

- void **set_enable_recursive_folding**(value: bool)
- bool **is_recursive_folding_enabled**()

If `true`, recursive folding is enabled for this **Tree**. Holding down `Shift` while clicking the fold arrow or using `ui_right`/`ui_left` shortcuts collapses or uncollapses the TreeItem and all its descendants.

bool **hide_folding** = `false`

- void **set_hide_folding**(value: bool)
- bool **is_folding_hidden**()

If `true`, the folding arrow is hidden.

bool **hide_root** = `false`

- void **set_hide_root**(value: bool)
- bool **is_root_hidden**()

If `true`, the tree's root is hidden.

bool **scroll_horizontal_enabled** = `true` 🔗

- void **set_h_scroll_enabled**(value: bool)
- bool **is_h_scroll_enabled**()

If `true`, enables horizontal scrolling.

bool **scroll_vertical_enabled** = `true` 🔗

- void **set_v_scroll_enabled**(value: bool)
- bool **is_v_scroll_enabled**()

If `true`, enables vertical scrolling.

SelectMode **select_mode** = `0` 🔗

- void **set_select_mode**(value: SelectMode)
- SelectMode **get_select_mode**()

Allows single or multiple selection. See the SelectMode constants.

Method Descriptions

void **clear**() 🔗

Clears the tree. This removes all items.

TreeItem **create_item**(parent: TreeItem = null, index: int = -1)

Creates an item in the tree and adds it as a child of `parent`, which can be either a valid TreeItem or `null`.

If `parent` is `null`, the root item will be the parent, or the new item will be the root itself if the tree is empty.

The new item will be the `index`-th child of parent, or it will be the last child if there are not enough siblings.

---

void **deselect_all**()

Deselects all tree items (rows and columns). In SELECT_MULTI mode also removes selection cursor.

---

bool **edit_selected**(force_edit: bool = false)

Edits the selected tree item as if it was clicked.

Either the item must be set editable with TreeItem.set_editable or `force_edit` must be `true`.

Returns `true` if the item could be edited. Fails if no item is selected.

---

void **ensure_cursor_is_visible**()

Makes the currently focused cell visible.

This will scroll the tree if necessary. In SELECT_ROW mode, this will not do horizontal scrolling, as all the cells in the selected row is focused logically.

Note: Despite the name of this method, the focus cursor itself is only visible in SELECT_MULTI mode.

---

int **get_button_id_at_position**(position: Vector2) const

Returns the button ID at `position`, or -1 if no button is there.

---

int **get_column_at_position**(position: Vector2) const

Returns the column index at `position`, or -1 if no item is there.

---

int **get_column_expand_ratio**(column: int) const

Returns the expand ratio assigned to the column.

String **get_column_title**(column: int) const

Returns the column's title.

HorizontalAlignment **get_column_title_alignment**(column: int) const

Returns the column title alignment.

TextDirection **get_column_title_direction**(column: int) const

Returns column title base writing direction.

String **get_column_title_language**(column: int) const

Returns column title language code.

---

int **get_column_width**(column: int) const 🔗

Returns the column's width in pixels.

Rect2 **get_custom_popup_rect**() const 🔗

Returns the rectangle for custom popups. Helper to create custom cell controls that display a popup. See TreeItem.set_cell_mode.

---

int **get_drop_section_at_position**(position: Vector2) const 🔗

Returns the drop section at `position`, or -100 if no item is there.

Values -1, 0, or 1 will be returned for the "above item", "on item", and "below item" drop sections, respectively. See DropModeFlags for a description of each drop section.

To get the item which the returned drop section is relative to, use get_item_at_position.

TreeItem **get_edited**() const 🔗

Returns the currently edited item. Can be used with item_edited to get the item that was modified.

GDScript

```
func _ready():
    $Tree.item_edited.connect(on_Tree_item_edited)

func on_Tree_item_edited():
    print($Tree.get_edited()) # This item just got edited (e.g. checked).
```

C#

```
public override void _Ready()
{
    GetNode<Tree>("Tree").ItemEdited += OnTreeItemEdited;
}

public void OnTreeItemEdited()
{
    GD.Print(GetNode<Tree>("Tree").GetEdited()); // This item just got edited (e.g. checked).
}
```

---

int **get_edited_column**() const

Returns the column for the currently edited item.

Rect2 **get_item_area_rect**(item: TreeItem, column: int = -1, button_index: int = -1) const

Returns the rectangle area for the specified TreeItem. If `column` is specified, only get the position and size of that column, otherwise get the rectangle containing all columns. If a button index is specified, the rectangle of that button will be returned.

TreeItem **get_item_at_position**(position: Vector2) const

Returns the tree item at the specified position (relative to the tree origin position).

TreeItem **get_next_selected**(from: TreeItem)

Returns the next selected TreeItem after the given one, or `null` if the end is reached.

If `from` is `null`, this returns the first selected item.

---

int **get_pressed_button**() const

Returns the last pressed button's index.

TreeItem **get_root**() const

Returns the tree's root item, or `null` if the tree is empty.

Vector2 **get_scroll**() const

Returns the current scrolling position.

TreeItem **get_selected**() const

Returns the currently focused item, or `null` if no item is focused.

In SELECT_ROW and SELECT_SINGLE modes, the focused item is same as the selected item. In SELECT_MULTI mode, the focused item is the item under the focus cursor, not necessarily selected.

To get the currently selected item(s), use get_next_selected.

---

int **get_selected_column**() const

Returns the currently focused column, or -1 if no column is focused.

In SELECT_SINGLE mode, the focused column is the selected column. In SELECT_ROW mode, the focused column is always 0 if any item is selected. In SELECT_MULTI mode, the focused column is the column under the focus cursor, and there are not necessarily any column selected.

To tell whether a column of an item is selected, use TreeItem.is_selected.

bool **is_column_clipping_content**(column: int) const

Returns `true` if the column has enabled clipping (see set_column_clip_content).

bool **is_column_expanding**(column: int) const

Returns `true` if the column has enabled expanding (see set_column_expand).

void **scroll_to_item**(item: TreeItem, center_on_item: bool = false)

Causes the **Tree** to jump to the specified TreeItem.

void **set_column_clip_content**(column: int, enable: bool)

Allows to enable clipping for column's content, making the content size ignored.

void **set_column_custom_minimum_width**(column: int, min_width: int)

Overrides the calculated minimum width of a column. It can be set to `0` to restore the default behavior. Columns that have the "Expand" flag will use their "min_width" in a similar fashion to Control.size_flags_stretch_ratio.

void **set_column_expand**(column: int, expand: bool)

If `true`, the column will have the "Expand" flag of Control. Columns that have the "Expand" flag will use their expand ratio in a similar fashion to Control.size_flags_stretch_ratio (see set_column_expand_ratio).

void **set_column_expand_ratio**(column: int, ratio: int)

Sets the relative expand ratio for a column. See set_column_expand.

void **set_column_title**(column: int, title: String)

Sets the title of a column.

void **set_column_title_alignment**(column: int, title_alignment: HorizontalAlignment)

Sets the column title alignment. Note that @GlobalScope.HORIZONTAL_ALIGNMENT_FILL is not supported for column titles.

void **set_column_title_direction**(column: int, direction: TextDirection)

Sets column title base writing direction.

void **set_column_title_language**(column: int, language: String)

Sets language code of column title used for line-breaking and text shaping algorithms, if left empty current locale is used instead.

void **set_selected**(item: TreeItem, column: int)

Selects the specified TreeItem and column.

Theme Property Descriptions

Color **children_hl_line_color** = `Color(0.27, 0.27, 0.27, 1)`

The Color of the relationship lines between the selected TreeItem and its children.

Color **custom_button_font_highlight** = `Color(0.95, 0.95, 0.95, 1)`

Text Color for a TreeItem.CELL_MODE_CUSTOM mode cell when it's hovered.

Color **drop_position_color** = `Color(1, 1, 1, 1)`

Color used to draw possible drop locations. See DropModeFlags constants for further description of drop locations.

Color **font_color** = `Color(0.7, 0.7, 0.7, 1)`

Default text Color of the item.

Color **font_disabled_color** = `Color(0.875, 0.875, 0.875, 0.5)`

Text Color for a TreeItem.CELL_MODE_CHECK mode cell when it's non-editable (see TreeItem.set_editable).

Color **font_outline_color** = `Color(0, 0, 0, 1)`

The tint of text outline of the item.

Color **font_selected_color** = `Color(1, 1, 1, 1)`

Text Color used when the item is selected.

Color **guide_color** = `Color(0.7, 0.7, 0.7, 0.25)`

Color of the guideline.

Color **parent_hl_line_color** = `Color(0.27, 0.27, 0.27, 1)`

The Color of the relationship lines between the selected TreeItem and its parents.

Color **relationship_line_color** = `Color(0.27, 0.27, 0.27, 1)`

The default Color of the relationship lines.

Color **title_button_color** = `Color(0.875, 0.875, 0.875, 1)`

Default text Color of the title button.

---

int **button_margin** = `4`

The horizontal space between each button in a cell.

---

int **children_hl_line_width** = `1`

The width of the relationship lines between the selected TreeItem and its children.

---

int **draw_guides** = `1`

Draws the guidelines if not zero, this acts as a boolean. The guideline is a horizontal line drawn at the bottom of each item.

---

int **draw_relationship_lines** = `0`

Draws the relationship lines if not zero, this acts as a boolean. Relationship lines are drawn at the start of child items to show hierarchy.

---

int **h_separation** = `4`

The horizontal space between item cells. This is also used as the margin at the start of an item when folding is disabled.

---

int **icon_max_width** = `0`

The maximum allowed width of the icon in item's cells. This limit is applied on top of the default size of the icon, but before the value set with TreeItem.set_icon_max_width. The height is adjusted according to the icon's ratio.

int **inner_item_margin_bottom** = `0` 🔗

The inner bottom margin of a cell.

int **inner_item_margin_left** = `0` 🔗

The inner left margin of a cell.

int **inner_item_margin_right** = `0` 🔗

The inner right margin of a cell.

int **inner_item_margin_top** = `0` 🔗

The inner top margin of a cell.

int **item_margin** = `16` 🔗

The horizontal margin at the start of an item. This is used when folding is enabled for the item.

int **outline_size** = `0` 🔗

The size of the text outline.

Note: If using a font with FontFile.multichannel_signed_distance_field enabled, its FontFile.msdf_pixel_range must be set to at least *twice* the value of outline_size for outline rendering to look correct. Otherwise, the outline may appear to be cut off earlier than intended.

---

int **parent_hl_line_margin** = `0`

The space between the parent relationship lines for the selected TreeItem and the relationship lines to its siblings that are not selected.

---

int **parent_hl_line_width** = `1`

The width of the relationship lines between the selected TreeItem and its parents.

---

int **relationship_line_width** = `1`

The default width of the relationship lines.

---

int **scroll_border** = `4`

The maximum distance between the mouse cursor and the control's border to trigger border scrolling when dragging.

---

int **scroll_speed** = `12`

The speed of border scrolling.

---

int **scrollbar_h_separation** = `4`

The horizontal separation of tree content and scrollbar.

---

int **scrollbar_margin_bottom** = `-1` 🔗

The bottom margin of the scrollbars. When negative, uses panel bottom margin.

---

int **scrollbar_margin_left** = `-1` 🔗

The left margin of the horizontal scrollbar. When negative, uses panel left margin.

---

int **scrollbar_margin_right** = `-1` 🔗

The right margin of the scrollbars. When negative, uses panel right margin.

---

int **scrollbar_margin_top** = `-1` 🔗

The top margin of the vertical scrollbar. When negative, uses panel top margin.

---

int **scrollbar_v_separation** = `4` 🔗

The vertical separation of tree content and scrollbar.

---

int **v_separation** = `4` 🔗

The vertical padding inside each item, i.e. the distance between the item's content and top/bottom border.

Font **font** 🔗

Font of the item's text.

Font **title_button_font** 🔗

Font of the title button's text.

int **font_size** 🔗

Font size of the item's text.

int **title_button_font_size** 🔗

Font size of the title button's text.

Texture2D **arrow** 🔗

The arrow icon used when a foldable item is not collapsed.

Texture2D **arrow_collapsed** 🔗

The arrow icon used when a foldable item is collapsed (for left-to-right layouts).

Texture2D **arrow_collapsed_mirrored** 🔗

The arrow icon used when a foldable item is collapsed (for right-to-left layouts).

Texture2D **checked** 🔗

The check icon to display when the TreeItem.CELL_MODE_CHECK mode cell is checked and editable (see TreeItem.set_editable).

Texture2D **checked_disabled**

The check icon to display when the TreeItem.CELL_MODE_CHECK mode cell is checked and non-editable (see TreeItem.set_editable).

Texture2D **indeterminate**

The check icon to display when the TreeItem.CELL_MODE_CHECK mode cell is indeterminate and editable (see TreeItem.set_editable).

Texture2D **indeterminate_disabled**

The check icon to display when the TreeItem.CELL_MODE_CHECK mode cell is indeterminate and non-editable (see TreeItem.set_editable).

Texture2D **select_arrow**

The arrow icon to display for the TreeItem.CELL_MODE_RANGE mode cell.

Texture2D **unchecked**

The check icon to display when the TreeItem.CELL_MODE_CHECK mode cell is unchecked and editable (see TreeItem.set_editable).

Texture2D **unchecked_disabled**

The check icon to display when the TreeItem.CELL_MODE_CHECK mode cell is unchecked and non-editable (see TreeItem.set_editable).

Texture2D **updown**

The updown arrow icon to display for the TreeItem.CELL_MODE_RANGE mode cell.

StyleBox **button_pressed**

StyleBox used when a button in the tree is pressed.

StyleBox **cursor**

StyleBox used for the cursor, when the **Tree** is being focused.

StyleBox **cursor_unfocused**

StyleBox used for the cursor, when the **Tree** is not being focused.

StyleBox **custom_button**

Default StyleBox for a TreeItem.CELL_MODE_CUSTOM mode cell when button is enabled with TreeItem.set_custom_as_button.

StyleBox **custom_button_hover**

StyleBox for a TreeItem.CELL_MODE_CUSTOM mode button cell when it's hovered.

StyleBox **custom_button_pressed** 🔗

StyleBox for a TreeItem.CELL_MODE_CUSTOM mode button cell when it's pressed.

StyleBox **focus** 🔗

The focused style for the **Tree**, drawn on top of everything.

StyleBox **panel** 🔗

The background style for the **Tree**.

StyleBox **selected** 🔗

StyleBox for the selected items, used when the **Tree** is not being focused.

StyleBox **selected_focus** 🔗

StyleBox for the selected items, used when the **Tree** is being focused.

StyleBox **title_button_hover** 🔗

StyleBox used when the title button is being hovered.

StyleBox **title_button_normal** 🔗

Default StyleBox for the title button.

StyleBox **title_button_pressed** 🔗

StyleBox used when the title button is being pressed.

VBoxContainer

Inherits: BoxContainer **<** Container **<** Control **<** CanvasItem **<** Node **<** Object

Inherited By: ColorPicker, FileSystemDock, ScriptEditorBase

A container that arranges its child controls vertically.

Description

A variant of BoxContainer that can only arrange its child controls vertically. Child controls are rearranged automatically when their minimum size changes.

Tutorials

- Using Containers
- 3D Voxel Demo [https://godotengine.org/asset-library/asset/2755]

VehicleBody3D

Inherits: RigidBody3D **<** PhysicsBody3D **<** CollisionObject3D **<** Node3D **<** Node **<** Object

A 3D physics body that simulates the behavior of a car.

Description

This physics body implements all the physics logic needed to simulate a car. It is based on the raycast vehicle system commonly found in physics engines. Aside from a CollisionShape3D for the main body of the vehicle, you must also add a VehicleWheel3D node for each wheel. You should also add a MeshInstance3D to this node for the 3D model of the vehicle, but this model should generally not include meshes for the wheels. You can control the vehicle by using the brake, engine_force, and steering properties. The position or orientation of this node shouldn't be changed directly.

Note: The origin point of your VehicleBody3D will determine the center of gravity of your vehicle. To make the vehicle more grounded, the origin point is usually kept low, moving the CollisionShape3D and MeshInstance3D upwards.

Note: This class has known issues and isn't designed to provide realistic 3D vehicle physics. If you want advanced vehicle physics, you may have to write your own physics integration using CharacterBody3D or RigidBody3D.

Tutorials

- 3D Truck Town Demo [https://godotengine.org/asset-library/asset/2752]

Properties

float	brake	`0.0`
float	engine_force	`0.0`
float	mass	`40.0` (overrides RigidBody3D)
float	steering	`0.0`

Property Descriptions

float **brake** = `0.0`

- void **set_brake**(value: float)
- float **get_brake**()

Slows down the vehicle by applying a braking force. The vehicle is only slowed down if the wheels are in contact with a surface. The force you need to apply to adequately slow down your vehicle depends on the RigidBody3D.mass of the vehicle. For a vehicle with a mass set to 1000, try a value in the 25 - 30 range for hard braking.

float **engine_force** = `0.0`

- void **set_engine_force**(value: float)
- float **get_engine_force**()

Accelerates the vehicle by applying an engine force. The vehicle is only sped up if the wheels that have VehicleWheel3D.use_as_traction set to `true` and are in contact with a surface. The RigidBody3D.mass of the vehicle has an effect on the acceleration of the vehicle. For a vehicle with a mass set to 1000, try a value in the 25 - 50 range for acceleration.

Note: The simulation does not take the effect of gears into account, you will need to add logic for this if you wish to simulate gears.

A negative value will result in the vehicle reversing.

---

float **steering** = `0.0` 🔗

- void **set_steering**(value: float)
- float **get_steering**()

The steering angle for the vehicle. Setting this to a non-zero value will result in the vehicle turning when it's moving. Wheels that have VehicleWheel3D.use_as_steering set to `true` will automatically be rotated.

Note: This property is edited in the inspector in degrees. In code the property is set in radians.

VehicleWheel3D

Inherits: Node3D **<** Node **<** Object

A 3D physics body for a VehicleBody3D that simulates the behavior of a wheel.

Description

A node used as a child of a VehicleBody3D parent to simulate the behavior of one of its wheels. This node also acts as a collider to detect if the wheel is touching a surface.

Note: This class has known issues and isn't designed to provide realistic 3D vehicle physics. If you want advanced vehicle physics, you may need to write your own physics integration using another PhysicsBody3D class.

Tutorials

- 3D Truck Town Demo [https://godotengine.org/asset-library/asset/2752]

Properties

float	brake	`0.0`
float	damping_compression	`0.83`

float	damping_relaxation	`0.88`
float	engine_force	`0.0`
float	steering	`0.0`
float	suspension_max_force	`6000.0`
float	suspension_stiffness	`5.88`
float	suspension_travel	`0.2`
bool	use_as_steering	`false`
bool	use_as_traction	`false`
float	wheel_friction_slip	`10.5`
float	wheel_radius	`0.5`
float	wheel_rest_length	`0.15`
float	wheel_roll_influence	`0.1`

Methods

Node3D	get_contact_body() const
float	get_rpm() const
float	get_skidinfo() const
bool	is_in_contact() const

Property Descriptions

float **brake** = `0.0` 🔗

- void **set_brake**(value: float)
- float **get_brake**()

Slows down the wheel by applying a braking force. The wheel is only slowed down if it is in contact with a surface. The force you need to apply to adequately slow down your vehicle depends on the RigidBody3D.mass of the vehicle. For a vehicle with a mass set to 1000, try a value in the 25 - 30 range for hard braking.

float **damping_compression** = `0.83` 🔗

- void **set_damping_compression**(value: float)
- float **get_damping_compression**()

The damping applied to the spring when the spring is being compressed. This value should be between 0.0 (no damping) and 1.0. A value of 0.0 means the car will keep bouncing as the spring keeps its energy. A good value for this is around 0.3 for a normal car, 0.5 for a race car.

---

float **damping_relaxation** = `0.88`

- void **set_damping_relaxation**(value: float)
- float **get_damping_relaxation**()

The damping applied to the spring when relaxing. This value should be between 0.0 (no damping) and 1.0. This value should always be slightly higher than the damping_compression property. For a damping_compression value of 0.3, try a relaxation value of 0.5.

---

float **engine_force** = `0.0`

- void **set_engine_force**(value: float)
- float **get_engine_force**()

Accelerates the wheel by applying an engine force. The wheel is only sped up if it is in contact with a surface. The RigidBody3D.mass of the vehicle has an effect on the acceleration of the vehicle. For a vehicle with a mass set to 1000, try a value in the 25 - 50 range for acceleration.

Note: The simulation does not take the effect of gears into account, you will need to add logic for this if you wish to simulate gears.

A negative value will result in the wheel reversing.

---

float **steering** = `0.0`

- void **set_steering**(value: float)
- float **get_steering**()

The steering angle for the wheel, in radians. Setting this to a non-zero value will result in the vehicle turning when it's moving.

---

float **suspension_max_force** = `6000.0` 🔗

- void **set_suspension_max_force**(value: float)
- float **get_suspension_max_force**()

The maximum force the spring can resist. This value should be higher than a quarter of the RigidBody3D.mass of the VehicleBody3D or the spring will not carry the weight of the vehicle. Good results are often obtained by a value that is about 3× to 4× this number.

---

float **suspension_stiffness** = `5.88` 🔗

- void **set_suspension_stiffness**(value: float)
- float **get_suspension_stiffness**()

This value defines the stiffness of the suspension. Use a value lower than 50 for an off-road car, a value between 50 and 100 for a race car and try something around 200 for something like a Formula 1 car.

---

float **suspension_travel** = `0.2` 🔗

- void **set_suspension_travel**(value: float)
- float **get_suspension_travel**()

This is the distance the suspension can travel. As Godot units are equivalent to meters, keep this setting relatively low. Try a value between 0.1 and 0.3 depending on the type of car.

---

bool **use_as_steering** = `false`

- void **set_use_as_steering**(value: bool)
- bool **is_used_as_steering**()

If `true`, this wheel will be turned when the car steers. This value is used in conjunction with VehicleBody3D.steering and ignored if you are using the per-wheel steering value instead.

---

bool **use_as_traction** = `false`

- void **set_use_as_traction**(value: bool)
- bool **is_used_as_traction**()

If `true`, this wheel transfers engine force to the ground to propel the vehicle forward. This value is used in conjunction with VehicleBody3D.engine_force and ignored if you are using the per-wheel engine_force value instead.

---

float **wheel_friction_slip** = `10.5`

- void **set_friction_slip**(value: float)
- float **get_friction_slip**()

This determines how much grip this wheel has. It is combined with the friction setting of the surface the wheel is in contact with. 0.0 means no grip, 1.0 is normal grip. For a drift car setup, try setting the grip of the rear wheels slightly lower than the front wheels, or use a lower value to simulate tire wear.

It's best to set this to 1.0 when starting out.

---

float **wheel_radius** = `0.5`

- void **set_radius**(value: float)
- float **get_radius**()

The radius of the wheel in meters.

float **wheel_rest_length** = `0.15` 🔗

- void **set_suspension_rest_length**(value: float)
- float **get_suspension_rest_length**()

This is the distance in meters the wheel is lowered from its origin point. Don't set this to 0.0 and move the wheel into position, instead move the origin point of your wheel (the gizmo in Godot) to the position the wheel will take when bottoming out, then use the rest length to move the wheel down to the position it should be in when the car is in rest.

float **wheel_roll_influence** = `0.1` 🔗

- void **set_roll_influence**(value: float)
- float **get_roll_influence**()

This value affects the roll of your vehicle. If set to 1.0 for all wheels, your vehicle will resist body roll, while a value of 0.0 will be prone to rolling over.

Method Descriptions

Node3D **get_contact_body**() const 🔗

Returns the contacting body node if valid in the tree, as Node3D. At the moment, GridMap is not supported so the node will be always of type PhysicsBody3D.

Returns `null` if the wheel is not in contact with a surface, or the contact body is not a PhysicsBody3D.

---

float **get_rpm**() const 🔗

Returns the rotational speed of the wheel in revolutions per minute.

---

float **get_skidinfo**() const 🔗

Returns a value between 0.0 and 1.0 that indicates whether this wheel is skidding. 0.0 is skidding (the wheel has lost grip, e.g. icy terrain), 1.0 means not skidding (the wheel has full grip, e.g. dry asphalt road).

---

bool **is_in_contact**() const 🔗

Returns `true` if this wheel is in contact with a surface.

VFlowContainer

Inherits: FlowContainer **<** Container **<** Control **<** CanvasItem **<** Node **<** Object

A container that arranges its child controls vertically and wraps them around at the borders.

Description

A variant of FlowContainer that can only arrange its child controls vertically, wrapping them around at the borders. This is similar to how text in a book wraps around when no more words can fit on a line, except vertically.

Tutorials

- Using Containers

VideoStreamPlayer

Inherits: Control **<** CanvasItem **<** Node **<** Object

A control used for video playback.

Description

A control used for playback of VideoStream resources.

Supported video formats are Ogg Theora [https://www.theora.org/] (`.ogv`, VideoStreamTheora) and any format exposed via a GDExtension plugin.

Warning: On Web, video playback *will* perform poorly due to missing architecture-specific assembly optimizations.

Tutorials

- Playing videos

Properties

int	audio_track	`0`
bool	autoplay	`false`
int	buffering_msec	`500`

StringName	bus	`&"Master"`
bool	expand	`false`
bool	loop	`false`
bool	paused	`false`
VideoStream	stream	
float	stream_position	
float	volume	
float	volume_db	`0.0`

Methods

float	get_stream_length() const
String	get_stream_name() const
Texture2D	get_video_texture() const

bool	is_playing() const
void	play()
void	stop()

Signals

finished()

Emitted when playback is finished.

Property Descriptions

int **audio_track** = `0`

- void **set_audio_track**(value: int)
- int **get_audio_track**()

The embedded audio track to play.

bool **autoplay** = `false`

- void **set_autoplay**(value: bool)
- bool **has_autoplay**()

If `true`, playback starts when the scene loads.

int **buffering_msec** = `500`

- void **set_buffering_msec**(value: int)
- int **get_buffering_msec**()

Amount of time in milliseconds to store in buffer while playing.

StringName **bus** = `&"Master"`

- void **set_bus**(value: StringName)
- StringName **get_bus**()

Audio bus to use for sound playback.

---

bool **expand** = `false`

- void **set_expand**(value: bool)
- bool **has_expand**()

If `true`, the video scales to the control size. Otherwise, the control minimum size will be automatically adjusted to match the video stream's dimensions.

---

bool **loop** = `false`

- void **set_loop**(value: bool)
- bool **has_loop**()

If `true`, the video restarts when it reaches its end.

---

bool **paused** = `false`

- void **set_paused**(value: bool)

- bool **is_paused**()

If `true`, the video is paused.

VideoStream **stream**

- void **set_stream**(value: VideoStream)
- VideoStream **get_stream**()

The assigned video stream. See description for supported formats.

---

float **stream_position**

- void **set_stream_position**(value: float)
- float **get_stream_position**()

The current position of the stream, in seconds.

Note: Changing this value won't have any effect as seeking is not implemented yet, except in video formats implemented by a GDExtension add-on.

---

float **volume**

- void **set_volume**(value: float)
- float **get_volume**()

Audio volume as a linear value.

---

float **volume_db** = `0.0`

- void **set_volume_db**(value: float)
- float **get_volume_db**()

Audio volume in dB.

Method Descriptions

float **get_stream_length**() const 🔗

The length of the current stream, in seconds.

Note: For VideoStreamTheora streams (the built-in format supported by Godot), this value will always be zero, as getting the stream length is not implemented yet. The feature may be supported by video formats implemented by a GDExtension add-on.

String **get_stream_name**() const 🔗

Returns the video stream's name, or `"<No Stream>"` if no video stream is assigned.

Texture2D **get_video_texture**() const 🔗

Returns the current frame as a Texture2D.

---

bool **is_playing**() const 🔗

Returns `true` if the video is playing.

Note: The video is still considered playing if paused during playback.

---

void **play**() 🔗

Starts the video playback from the beginning. If the video is paused, this will not unpause the video.

---

void **stop**() 🔗

Stops the video playback and sets the stream position to 0.

Note: Although the stream position will be set to 0, the first frame of the video stream won't become the current frame.

Viewport

Inherits: Node < Object

Inherited By: SubViewport, Window

Abstract base class for viewports. Encapsulates drawing and interaction with a game world.

Description

A **Viewport** creates a different view into the screen, or a sub-view inside another viewport. Child 2D nodes will display on it, and child Camera3D 3D nodes will render on it too.

Optionally, a viewport can have its own 2D or 3D world, so it doesn't share what it draws with other viewports.

Viewports can also choose to be audio listeners, so they generate positional audio depending on a 2D or 3D camera child of it.

Also, viewports can be assigned to different screens in case the devices have multiple screens.

Finally, viewports can also behave as render targets, in which case they will not be visible unless the associated texture is used to draw.

Tutorials

- Using Viewports
- Viewport and canvas transforms
- GUI in 3D Viewport Demo [https://godotengine.org/asset-library/asset/2807]
- 3D in 2D Viewport Demo [https://godotengine.org/asset-library/asset/2804]
- 2D in 3D Viewport Demo [https://godotengine.org/asset-library/asset/2803]
- Screen Capture Demo [https://godotengine.org/asset-library/asset/2808]
- Dynamic Split Screen Demo [https://godotengine.org/asset-library/asset/2806]
- 3D Resolution Scaling Demo [https://godotengine.org/asset-library/asset/2805]

Properties

bool	audio_listener_enable_2d	`false`
bool	audio_listener_enable_3d	`false`
int	canvas_cull_mask	`4294967295`

DefaultCanvasItemTextureFilter	canvas_item_default_texture_filter	`1`
DefaultCanvasItemTextureRepeat	canvas_item_default_texture_repeat	`0`
Transform2D	canvas_transform	
DebugDraw	debug_draw	`0`
bool	disable_3d	`false`
float	fsr_sharpness	`0.2`
Transform2D	global_canvas_transform	
bool	gui_disable_input	`false`
bool	gui_embed_subwindows	`false`
bool	gui_snap_controls_to_pixels	`true`
bool	handle_input_locally	`true`
float	mesh_lod_threshold	`1.0`
MSAA	msaa_2d	`0`
MSAA	msaa_3d	`0`
bool	own_world_3d	`false`
bool	physics_object_picking	`false`
bool	physics_object_picking_first_only	`false`
bool	physics_object_picking_sort	`false`

bool	positional_shadow_atlas_16_bits	`true`
PositionalShadowAtlasQuadrantSubdiv	positional_shadow_atlas_quad_0	`2`
PositionalShadowAtlasQuadrantSubdiv	positional_shadow_atlas_quad_1	`2`
PositionalShadowAtlasQuadrantSubdiv	positional_shadow_atlas_quad_2	`3`
PositionalShadowAtlasQuadrantSubdiv	positional_shadow_atlas_quad_3	`4`
int	positional_shadow_atlas_size	`2048`
Scaling3DMode	scaling_3d_mode	`0`
float	scaling_3d_scale	`1.0`
ScreenSpaceAA	screen_space_aa	`0`
SDFOversize	sdf_oversize	`1`
SDFScale	sdf_scale	`1`
bool	snap_2d_transforms_to_pixel	`false`
bool	snap_2d_vertices_to_pixel	`false`
float	texture_mipmap_bias	`0.0`
bool	transparent_bg	`false`
bool	use_debanding	`false`
bool	use_hdr_2d	`false`

bool	use_occlusion_culling	`false`
bool	use_taa	`false`
bool	use_xr	`false`
VRSMode	vrs_mode	`0`
Texture2D	vrs_texture	
VRSUpdateMode	vrs_update_mode	`1`
World2D	world_2d	
World3D	world_3d	

Methods

World2D	find_world_2d() const
World3D	find_world_3d() const
Camera2D	get_camera_2d() const
Camera3D	get_camera_3d() const
bool	get_canvas_cull_mask_bit(layer: int) const
Array[Window]	get_embedded_subwindows() const
Transform2D	get_final_transform() const
Vector2	get_mouse_position() const

PositionalShadowAtlasQuadrantSubdiv	get_positional_shadow_atlas_quadrant_subdiv(quadrant: int) const
int	get_render_info(type: RenderInfoType, info: RenderInfo)
Transform2D	get_screen_transform() const
ViewportTexture	get_texture() const
RID	get_viewport_rid() const
Rect2	get_visible_rect() const
Variant	gui_get_drag_data() const
Control	gui_get_focus_owner() const
Control	gui_get_hovered_control() const
bool	gui_is_drag_successful() const
bool	gui_is_dragging() const
void	gui_release_focus()
bool	is_input_handled() const
void	push_input(event: InputEvent, in_local_coords: bool = false)
void	push_text_input(text: String)
void	push_unhandled_input(event: InputEvent, in_local_coords: bool = false)

void	set_canvas_cull_mask_bit(layer: int, enable: bool)
void	set_input_as_handled()
void	set_positional_shadow_atlas_quadrant_subdiv(quadrant: int, subdiv: PositionalShadowAtlasQuadrantSubdiv)
void	update_mouse_cursor_state()
void	warp_mouse(position: Vector2)

Signals

gui_focus_changed(node: Control) 🔗

Emitted when a Control node grabs keyboard focus.

Note: A Control node losing focus doesn't cause this signal to be emitted.

---

size_changed() 🔗

Emitted when the size of the viewport is changed, whether by resizing of window, or some other means.

Enumerations

enum **PositionalShadowAtlasQuadrantSubdiv**: 🔗

PositionalShadowAtlasQuadrantSubdiv **SHADOW_ATLAS_QUADRANT_SUBDIV_DISABLED** = `0`

This quadrant will not be used.

PositionalShadowAtlasQuadrantSubdiv **SHADOW_ATLAS_QUADRANT_SUBDIV_1** = `1`

This quadrant will only be used by one shadow map.

PositionalShadowAtlasQuadrantSubdiv **SHADOW_ATLAS_QUADRANT_SUBDIV_4** = `2`

This quadrant will be split in 4 and used by up to 4 shadow maps.

PositionalShadowAtlasQuadrantSubdiv **SHADOW_ATLAS_QUADRANT_SUBDIV_16** = `3`

This quadrant will be split 16 ways and used by up to 16 shadow maps.

PositionalShadowAtlasQuadrantSubdiv **SHADOW_ATLAS_QUADRANT_SUBDIV_64** = `4`

This quadrant will be split 64 ways and used by up to 64 shadow maps.

PositionalShadowAtlasQuadrantSubdiv **SHADOW_ATLAS_QUADRANT_SUBDIV_256** = `5`

This quadrant will be split 256 ways and used by up to 256 shadow maps. Unless the positional_shadow_atlas_size is very high, the shadows in this quadrant will be very low resolution.

PositionalShadowAtlasQuadrantSubdiv **SHADOW_ATLAS_QUADRANT_SUBDIV_1024** = `6`

This quadrant will be split 1024 ways and used by up to 1024 shadow maps. Unless the positional_shadow_atlas_size is very high, the shadows in this quadrant will be very low resolution.

PositionalShadowAtlasQuadrantSubdiv **SHADOW_ATLAS_QUADRANT_SUBDIV_MAX** = `7`

Represents the size of the PositionalShadowAtlasQuadrantSubdiv enum.

---

enum **Scaling3DMode**:

Scaling3DMode **SCALING_3D_MODE_BILINEAR** = `0`

Use bilinear scaling for the viewport's 3D buffer. The amount of scaling can be set using scaling_3d_scale. Values less than `1.0` will result in undersampling while values greater than `1.0` will result in supersampling. A value of `1.0` disables scaling.

Scaling3DMode **SCALING_3D_MODE_FSR** = `1`

Use AMD FidelityFX Super Resolution 1.0 upscaling for the viewport's 3D buffer. The amount of scaling can be set using scaling_3d_scale. Values less than `1.0` will be result in the viewport being upscaled using FSR. Values greater than `1.0` are not supported and bilinear downsampling will be used instead. A value of `1.0` disables scaling.

Scaling3DMode **SCALING_3D_MODE_FSR2** = `2`

Use AMD FidelityFX Super Resolution 2.2 upscaling for the viewport's 3D buffer. The amount of scaling can be set using scaling_3d_scale. Values less than `1.0` will be result in the viewport being upscaled using FSR2. Values greater than `1.0` are not supported and bilinear downsampling will be used instead. A value of `1.0` will use FSR2 at native resolution as a TAA solution.

Scaling3DMode **SCALING_3D_MODE_MAX** = `3`

Represents the size of the Scaling3DMode enum.

---

enum **MSAA**:

MSAA **MSAA_DISABLED** = `0`

Multisample antialiasing mode disabled. This is the default value, and is also the fastest setting.

MSAA **MSAA_2X** = `1`

Use 2× Multisample Antialiasing. This has a moderate performance cost. It helps reduce aliasing noticeably, but 4× MSAA still looks substantially better.

MSAA **MSAA_4X** = `2`

Use 4× Multisample Antialiasing. This has a significant performance cost, and is generally a good compromise between performance and quality.

MSAA **MSAA_8X** = `3`

Use 8× Multisample Antialiasing. This has a very high performance cost. The difference between 4× and 8× MSAA may not always be visible in real gameplay conditions. Likely unsupported on low-end and older hardware.

MSAA **MSAA_MAX** = `4`

Represents the size of the MSAA enum.

---

enum **ScreenSpaceAA**:

ScreenSpaceAA **SCREEN_SPACE_AA_DISABLED** = `0`

Do not perform any antialiasing in the full screen post-process.

ScreenSpaceAA **SCREEN_SPACE_AA_FXAA** = `1`

Use fast approximate antialiasing. FXAA is a popular screen-space antialiasing method, which is fast but will make the image look blurry, especially at lower resolutions. It can still work relatively well at large resolutions such as 1440p and 4K.

ScreenSpaceAA **SCREEN_SPACE_AA_MAX** = `2`

Represents the size of the ScreenSpaceAA enum.

---

enum **RenderInfo**:

RenderInfo **RENDER_INFO_OBJECTS_IN_FRAME** = `0`

Amount of objects in frame.

RenderInfo **RENDER_INFO_PRIMITIVES_IN_FRAME** = `1`

Amount of vertices in frame.

RenderInfo **RENDER_INFO_DRAW_CALLS_IN_FRAME** = `2`

Amount of draw calls in frame.

RenderInfo **RENDER_INFO_MAX** = `3`

Represents the size of the RenderInfo enum.

---

enum **RenderInfoType**: 🔗

RenderInfoType **RENDER_INFO_TYPE_VISIBLE** = `0`

Visible render pass (excluding shadows).

RenderInfoType **RENDER_INFO_TYPE_SHADOW** = `1`

Shadow render pass. Objects will be rendered several times depending on the number of amounts of lights with shadows and the number of directional shadow splits.

RenderInfoType **RENDER_INFO_TYPE_CANVAS** = `2`

Canvas item rendering. This includes all 2D rendering.

RenderInfoType **RENDER_INFO_TYPE_MAX** = `3`

Represents the size of the RenderInfoType enum.

---

enum **DebugDraw**: 🔗

DebugDraw **DEBUG_DRAW_DISABLED** = `0`

Objects are displayed normally.

DebugDraw **DEBUG_DRAW_UNSHADED** = `1`

Objects are displayed without light information.

DebugDraw **DEBUG_DRAW_LIGHTING** = `2`

Objects are displayed without textures and only with lighting information.

DebugDraw **DEBUG_DRAW_OVERDRAW** = `3`

Objects are displayed semi-transparent with additive blending so you can see where they are drawing over top of one another. A higher overdraw means you are wasting performance on drawing pixels that are being hidden behind others.

DebugDraw **DEBUG_DRAW_WIREFRAME** = `4`

Objects are displayed as wireframe models.

DebugDraw **DEBUG_DRAW_NORMAL_BUFFER** = `5`

Objects are displayed without lighting information and their textures replaced by normal mapping.

DebugDraw **DEBUG_DRAW_VOXEL_GI_ALBEDO** = `6`

Objects are displayed with only the albedo value from VoxelGIs.

DebugDraw **DEBUG_DRAW_VOXEL_GI_LIGHTING** = `7`

Objects are displayed with only the lighting value from VoxelGIs.

DebugDraw **DEBUG_DRAW_VOXEL_GI_EMISSION** = `8`

Objects are displayed with only the emission color from VoxelGIs.

DebugDraw **DEBUG_DRAW_SHADOW_ATLAS** = `9`

Draws the shadow atlas that stores shadows from OmniLight3Ds and SpotLight3Ds in the upper left quadrant of the **Viewport**.

DebugDraw **DEBUG_DRAW_DIRECTIONAL_SHADOW_ATLAS** = `10`

Draws the shadow atlas that stores shadows from DirectionalLight3Ds in the upper left quadrant of the **Viewport**.

DebugDraw **DEBUG_DRAW_SCENE_LUMINANCE** = `11`

Draws the scene luminance buffer (if available) in the upper left quadrant of the **Viewport**.

DebugDraw **DEBUG_DRAW_SSAO** = `12`

Draws the screen-space ambient occlusion texture instead of the scene so that you can clearly see how it is affecting objects. In order for this display mode to work, you must have Environment.ssao_enabled set in your WorldEnvironment.

DebugDraw **DEBUG_DRAW_SSIL** = `13`

Draws the screen-space indirect lighting texture instead of the scene so that you can clearly see how it is affecting objects. In order for this display mode to work, you must have Environment.ssil_enabled set in your WorldEnvironment.

DebugDraw **DEBUG_DRAW_PSSM_SPLITS** = `14`

Colors each PSSM split for the DirectionalLight3Ds in the scene a different color so you can see where the splits are. In order, they will be colored red, green, blue, and yellow.

DebugDraw **DEBUG_DRAW_DECAL_ATLAS** = `15`

Draws the decal atlas used by Decals and light projector textures in the upper left quadrant of the **Viewport**.

DebugDraw **DEBUG_DRAW_SDFGI** = `16`

Draws the cascades used to render signed distance field global illumination (SDFGI).

Does nothing if the current environment's Environment.sdfgi_enabled is `false` or SDFGI is not supported on the platform.

DebugDraw **DEBUG_DRAW_SDFGI_PROBES** = `17`

Draws the probes used for signed distance field global illumination (SDFGI).

Does nothing if the current environment's Environment.sdfgi_enabled is `false` or SDFGI is not supported on the platform.

DebugDraw **DEBUG_DRAW_GI_BUFFER** = `18`

Draws the buffer used for global illumination (GI).

DebugDraw **DEBUG_DRAW_DISABLE_LOD** = `19`

Draws all of the objects at their highest polycount, without low level of detail (LOD).

DebugDraw **DEBUG_DRAW_CLUSTER_OMNI_LIGHTS** = `20`

Draws the cluster used by OmniLight3D nodes to optimize light rendering.

DebugDraw **DEBUG_DRAW_CLUSTER_SPOT_LIGHTS** = `21`

Draws the cluster used by SpotLight3D nodes to optimize light rendering.

DebugDraw **DEBUG_DRAW_CLUSTER_DECALS** = `22`

Draws the cluster used by Decal nodes to optimize decal rendering.

DebugDraw **DEBUG_DRAW_CLUSTER_REFLECTION_PROBES** = `23`

Draws the cluster used by ReflectionProbe nodes to optimize decal rendering.

DebugDraw **DEBUG_DRAW_OCCLUDERS** = `24`

Draws the buffer used for occlusion culling.

DebugDraw **DEBUG_DRAW_MOTION_VECTORS** = `25`

Draws vector lines over the viewport to indicate the movement of pixels between frames.

DebugDraw **DEBUG_DRAW_INTERNAL_BUFFER** = `26`

Draws the internal resolution buffer of the scene before post-processing is applied.

---

enum **DefaultCanvasItemTextureFilter**: 🔗

DefaultCanvasItemTextureFilter **DEFAULT_CANVAS_ITEM_TEXTURE_FILTER_NEAREST** = `0`

The texture filter reads from the nearest pixel only. This makes the texture look pixelated from up close, and grainy from a distance (due to mipmaps not being sampled).

DefaultCanvasItemTextureFilter **DEFAULT_CANVAS_ITEM_TEXTURE_FILTER_LINEAR** = `1`

The texture filter blends between the nearest 4 pixels. This makes the texture look smooth from up close, and grainy from a distance (due to mipmaps not being sampled).

DefaultCanvasItemTextureFilter **DEFAULT_CANVAS_ITEM_TEXTURE_FILTER_LINEAR_WITH_MIPMAPS** = `2`

The texture filter blends between the nearest 4 pixels and between the nearest 2 mipmaps (or uses the nearest mipmap if ProjectSettings.rendering/textures/default_filters/use_nearest_mipmap_filter is `true`). This makes the texture look smooth from up close, and smooth from a distance.

Use this for non-pixel art textures that may be viewed at a low scale (e.g. due to Camera2D zoom or sprite scaling), as mipmaps are important to smooth out pixels that are smaller than on-screen pixels.

DefaultCanvasItemTextureFilter **DEFAULT_CANVAS_ITEM_TEXTURE_FILTER_NEAREST_WITH_MIPMAPS** = `3`

The texture filter reads from the nearest pixel and blends between the nearest 2 mipmaps (or uses the nearest mipmap if ProjectSettings.rendering/textures/default_filters/use_nearest_mipmap_filter is `true`). This makes the texture look pixelated from up close, and smooth from a distance.

Use this for non-pixel art textures that may be viewed at a low scale (e.g. due to Camera2D zoom or sprite scaling), as mipmaps are important to smooth out pixels that are smaller than on-screen pixels.

DefaultCanvasItemTextureFilter **DEFAULT_CANVAS_ITEM_TEXTURE_FILTER_MAX** = `4`

Represents the size of the DefaultCanvasItemTextureFilter enum.

---

enum **DefaultCanvasItemTextureRepeat**: 🔗

DefaultCanvasItemTextureRepeat **DEFAULT_CANVAS_ITEM_TEXTURE_REPEAT_DISABLED** = `0`

Disables textures repeating. Instead, when reading UVs outside the 0-1 range, the value will be clamped to the edge of the texture, resulting in a stretched out look at the borders of the texture.

DefaultCanvasItemTextureRepeat **DEFAULT_CANVAS_ITEM_TEXTURE_REPEAT_ENABLED** = `1`

Enables the texture to repeat when UV coordinates are outside the 0-1 range. If using one of the linear filtering modes, this can result in artifacts at the edges of a texture when the sampler filters across the edges of the texture.

DefaultCanvasItemTextureRepeat **DEFAULT_CANVAS_ITEM_TEXTURE_REPEAT_MIRROR** = `2`

Flip the texture when repeating so that the edge lines up instead of abruptly changing.

DefaultCanvasItemTextureRepeat **DEFAULT_CANVAS_ITEM_TEXTURE_REPEAT_MAX** = `3`

Represents the size of the DefaultCanvasItemTextureRepeat enum.

---

enum **SDFOversize**:

SDFOversize **SDF_OVERSIZE_100_PERCENT** = `0`

The signed distance field only covers the viewport's own rectangle.

SDFOversize **SDF_OVERSIZE_120_PERCENT** = `1`

The signed distance field is expanded to cover 20% of the viewport's size around the borders.

SDFOversize **SDF_OVERSIZE_150_PERCENT** = `2`

The signed distance field is expanded to cover 50% of the viewport's size around the borders.

SDFOversize **SDF_OVERSIZE_200_PERCENT** = `3`

The signed distance field is expanded to cover 100% (double) of the viewport's size around the borders.

SDFOversize **SDF_OVERSIZE_MAX** = `4`

Represents the size of the SDFOversize enum.

---

enum **SDFScale**:

SDFScale **SDF_SCALE_100_PERCENT** = `0`

The signed distance field is rendered at full resolution.

SDFScale **SDF_SCALE_50_PERCENT** = `1`

The signed distance field is rendered at half the resolution of this viewport.

SDFScale **SDF_SCALE_25_PERCENT** = `2`

The signed distance field is rendered at a quarter the resolution of this viewport.

SDFScale **SDF_SCALE_MAX** = `3`

Represents the size of the SDFScale enum.

---

enum **VRSMode**:

VRSMode **VRS_DISABLED** = `0`

Variable Rate Shading is disabled.

VRSMode **VRS_TEXTURE** = `1`

Variable Rate Shading uses a texture. Note, for stereoscopic use a texture atlas with a texture for each view.

VRSMode **VRS_XR** = `2`

Variable Rate Shading's texture is supplied by the primary XRInterface.

VRSMode **VRS_MAX** = `3`

Represents the size of the VRSMode enum.

---

enum **VRSUpdateMode**:

VRSUpdateMode **VRS_UPDATE_DISABLED** = `0`

The input texture for variable rate shading will not be processed.

VRSUpdateMode **VRS_UPDATE_ONCE** = `1`

The input texture for variable rate shading will be processed once.

VRSUpdateMode **VRS_UPDATE_ALWAYS** = `2`

The input texture for variable rate shading will be processed each frame.

VRSUpdateMode **VRS_UPDATE_MAX** = `3`

Represents the size of the VRSUpdateMode enum.

Property Descriptions

bool **audio_listener_enable_2d** = `false`

- void **set_as_audio_listener_2d**(value: bool)
- bool **is_audio_listener_2d**()

If `true`, the viewport will process 2D audio streams.

---

bool **audio_listener_enable_3d** = `false`

- void **set_as_audio_listener_3d**(value: bool)

- bool **is_audio_listener_3d**()

If `true`, the viewport will process 3D audio streams.

---

int **canvas_cull_mask** = `4294967295`

- void **set_canvas_cull_mask**(value: int)
- int **get_canvas_cull_mask**()

The rendering layers in which this **Viewport** renders CanvasItem nodes.

DefaultCanvasItemTextureFilter **canvas_item_default_texture_filter** = `1`

- void **set_default_canvas_item_texture_filter**(value: DefaultCanvasItemTextureFilter)
- DefaultCanvasItemTextureFilter **get_default_canvas_item_texture_filter**()

Sets the default filter mode used by CanvasItems in this Viewport. See DefaultCanvasItemTextureFilter for options.

DefaultCanvasItemTextureRepeat **canvas_item_default_texture_repeat** = `0`

- void **set_default_canvas_item_texture_repeat**(value: DefaultCanvasItemTextureRepeat)
- DefaultCanvasItemTextureRepeat **get_default_canvas_item_texture_repeat**()

Sets the default repeat mode used by CanvasItems in this Viewport. See DefaultCanvasItemTextureRepeat for options.

Transform2D **canvas_transform**

- void **set_canvas_transform**(value: Transform2D)
- Transform2D **get_canvas_transform**()

The canvas transform of the viewport, useful for changing the on-screen positions of all child CanvasItems. This is relative to the global canvas transform of the viewport.

DebugDraw **debug_draw** = `0`

- void **set_debug_draw**(value: DebugDraw)
- DebugDraw **get_debug_draw**()

The overlay mode for test rendered geometry in debug purposes.

---

bool **disable_3d** = `false`

- void **set_disable_3d**(value: bool)

- bool **is_3d_disabled**()

Disable 3D rendering (but keep 2D rendering).

---

float **fsr_sharpness** = `0.2`

- void **set_fsr_sharpness**(value: float)
- float **get_fsr_sharpness**()

Determines how sharp the upscaled image will be when using the FSR upscaling mode. Sharpness halves with every whole number. Values go from 0.0 (sharpest) to 2.0. Values above 2.0 won't make a visible difference.

To control this property on the root viewport, set the ProjectSettings.rendering/scaling_3d/fsr_sharpness project setting.

Transform2D **global_canvas_transform**

- void **set_global_canvas_transform**(value: Transform2D)
- Transform2D **get_global_canvas_transform**()

The global canvas transform of the viewport. The canvas transform is relative to this.

---

bool **gui_disable_input** = `false`

- void **set_disable_input**(value: bool)
- bool **is_input_disabled**()

If `true`, the viewport will not receive input events.

---

bool **gui_embed_subwindows** = `false`

- void **set_embedding_subwindows**(value: bool)
- bool **is_embedding_subwindows**()

If `true`, sub-windows (popups and dialogs) will be embedded inside application window as control-like nodes. If `false`, they will appear as separate windows handled by the operating system.

---

bool **gui_snap_controls_to_pixels** = `true`

- void **set_snap_controls_to_pixels**(value: bool)
- bool **is_snap_controls_to_pixels_enabled**()

If `true`, the GUI controls on the viewport will lay pixel perfectly.

---

bool **handle_input_locally** = `true`

- void **set_handle_input_locally**(value: bool)
- bool **is_handling_input_locally**()

If `true`, this viewport will mark incoming input events as handled by itself. If `false`, this is instead done by the first parent viewport that is set to handle input locally.

A SubViewportContainer will automatically set this property to `false` for the **Viewport** contained inside of it.

See also set_input_as_handled and is_input_handled.

---

float **mesh_lod_threshold** = `1.0`

- void **set_mesh_lod_threshold**(value: float)
- float **get_mesh_lod_threshold**()

The automatic LOD bias to use for meshes rendered within the **Viewport** (this is analogous to ReflectionProbe.mesh_lod_threshold). Higher values will use less detailed versions of meshes that have LOD variations generated. If set to `0.0`, automatic LOD is disabled. Increase mesh_lod_threshold to improve performance at the cost of geometry detail.

To control this property on the root viewport, set the ProjectSettings.rendering/mesh_lod/lod_change/threshold_pixels project setting.

Note: mesh_lod_threshold does not affect GeometryInstance3D visibility ranges (also known as "manual" LOD or hierarchical LOD).

MSAA **msaa_2d** = `0`

- void **set_msaa_2d**(value: MSAA)
- MSAA **get_msaa_2d**()

The multisample anti-aliasing mode for 2D/Canvas rendering. A higher number results in smoother edges at the cost of significantly worse performance. A value of 2 or 4 is best unless targeting very high-end systems. This has no effect on shader-induced aliasing or texture aliasing.

MSAA **msaa_3d** = `0`

- void **set_msaa_3d**(value: MSAA)
- MSAA **get_msaa_3d**()

The multisample anti-aliasing mode for 3D rendering. A higher number results in smoother edges at the cost of significantly worse performance. A value of 2 or 4 is best unless targeting very high-end systems. See also bilinear scaling 3d scaling_3d_mode for supersampling, which provides higher quality but is much more expensive. This has no effect on shader-induced aliasing or texture aliasing.

bool **own_world_3d** = `false`

- void **set_use_own_world_3d**(value: bool)
- bool **is_using_own_world_3d**()

If `true`, the viewport will use a unique copy of the World3D defined in world_3d.

bool **physics_object_picking** = `false`

- void **set_physics_object_picking**(value: bool)
- bool **get_physics_object_picking**()

If `true`, the objects rendered by viewport become subjects of mouse picking process.

Note: The number of simultaneously pickable objects is limited to 64 and they are selected in a non-deterministic order, which can be different in each picking process.

bool **physics_object_picking_first_only** = `false`

- void **set_physics_object_picking_first_only**(value: bool)
- bool **get_physics_object_picking_first_only**()

If `true`, the input_event signal will only be sent to one physics object in the mouse picking process. If you want to get the top object only, you must also enable physics_object_picking_sort.

If `false`, an input_event signal will be sent to all physics objects in the mouse picking process.

This applies to 2D CanvasItem object picking only.

bool **physics_object_picking_sort** = `false`

- void **set_physics_object_picking_sort**(value: bool)
- bool **get_physics_object_picking_sort**()

If `true`, objects receive mouse picking events sorted primarily by their CanvasItem.z_index and secondarily by their position in the scene tree. If `false`, the order is undetermined.

Note: This setting is disabled by default because of its potential expensive computational cost.

Note: Sorting happens after selecting the pickable objects. Because of the limitation of 64 simultaneously pickable objects, it is not guaranteed that the object with the highest CanvasItem.z_index receives the picking event.

bool **positional_shadow_atlas_16_bits** = `true`

- void **set_positional_shadow_atlas_16_bits**(value: bool)

- bool **get_positional_shadow_atlas_16_bits**()

Use 16 bits for the omni/spot shadow depth map. Enabling this results in shadows having less precision and may result in shadow acne, but can lead to performance improvements on some devices.

PositionalShadowAtlasQuadrantSubdiv **positional_shadow_atlas_quad_0** = `2`

- void **set_positional_shadow_atlas_quadrant_subdiv**(quadrant: int, subdiv: PositionalShadowAtlasQuadrantSubdiv)
- PositionalShadowAtlasQuadrantSubdiv **get_positional_shadow_atlas_quadrant_subdiv**(quadrant: int) const

The subdivision amount of the first quadrant on the shadow atlas.

PositionalShadowAtlasQuadrantSubdiv **positional_shadow_atlas_quad_1** = `2`

- void **set_positional_shadow_atlas_quadrant_subdiv**(quadrant: int, subdiv: PositionalShadowAtlasQuadrantSubdiv)
- PositionalShadowAtlasQuadrantSubdiv **get_positional_shadow_atlas_quadrant_subdiv**(quadrant: int) const

The subdivision amount of the second quadrant on the shadow atlas.

PositionalShadowAtlasQuadrantSubdiv **positional_shadow_atlas_quad_2** = `3`

- void **set_positional_shadow_atlas_quadrant_subdiv**(quadrant: int, subdiv: PositionalShadowAtlasQuadrantSubdiv)
- PositionalShadowAtlasQuadrantSubdiv **get_positional_shadow_atlas_quadrant_subdiv**(quadrant: int) const

The subdivision amount of the third quadrant on the shadow atlas.

PositionalShadowAtlasQuadrantSubdiv **positional_shadow_atlas_quad_3** = `4`

- void **set_positional_shadow_atlas_quadrant_subdiv**(quadrant: int, subdiv: PositionalShadowAtlasQuadrantSubdiv)
- PositionalShadowAtlasQuadrantSubdiv **get_positional_shadow_atlas_quadrant_subdiv**(quadrant: int) const

The subdivision amount of the fourth quadrant on the shadow atlas.

---

int **positional_shadow_atlas_size** = `2048`

- void **set_positional_shadow_atlas_size**(value: int)
- int **get_positional_shadow_atlas_size**()

The shadow atlas' resolution (used for omni and spot lights). The value is rounded up to the nearest power of 2.

Note: If this is set to `0`, no positional shadows will be visible at all. This can improve performance significantly on low-end systems by reducing both the CPU and GPU load (as fewer draw calls are needed to draw the scene without shadows).

Scaling3DMode **scaling_3d_mode** = `0` 🔗

- void **set_scaling_3d_mode**(value: Scaling3DMode)
- Scaling3DMode **get_scaling_3d_mode**()

Sets scaling 3d mode. Bilinear scaling renders at different resolution to either undersample or supersample the viewport. FidelityFX Super Resolution 1.0, abbreviated to FSR, is an upscaling technology that produces high quality images at fast framerates by using a spatially aware upscaling algorithm. FSR is slightly more expensive than bilinear, but it produces significantly higher image quality. FSR should be used where possible.

To control this property on the root viewport, set the ProjectSettings.rendering/scaling_3d/mode project setting.

---

float **scaling_3d_scale** = `1.0` 🔗

- void **set_scaling_3d_scale**(value: float)
- float **get_scaling_3d_scale**()

Scales the 3D render buffer based on the viewport size uses an image filter specified in ProjectSettings.rendering/scaling_3d/mode to scale the output image to the full viewport size. Values lower than `1.0` can be used to speed up 3D rendering at the cost of quality (undersampling). Values greater than `1.0` are only valid for bilinear mode and can be used to improve 3D rendering quality at a high performance cost (supersampling). See also ProjectSettings.rendering/anti_aliasing/quality/msaa_3d for multi-sample antialiasing, which is significantly cheaper but only smooths the edges of polygons.

When using FSR upscaling, AMD recommends exposing the following values as preset options to users "Ultra Quality: 0.77", "Quality: 0.67", "Balanced: 0.59", "Performance: 0.5" instead of exposing the entire scale.

To control this property on the root viewport, set the ProjectSettings.rendering/scaling_3d/scale project setting.

ScreenSpaceAA **screen_space_aa** = `0` 🔗

- void **set_screen_space_aa**(value: ScreenSpaceAA)
- ScreenSpaceAA **get_screen_space_aa**()

Sets the screen-space antialiasing method used. Screen-space antialiasing works by selectively blurring edges in a post-process shader. It differs from MSAA which takes multiple coverage samples while rendering objects. Screen-space AA methods are typically faster than MSAA and will smooth out specular aliasing, but tend to make scenes appear blurry.

SDFOversize **sdf_oversize** = `1` 🔗

- void **set_sdf_oversize**(value: SDFOversize)
- SDFOversize **get_sdf_oversize**()

Controls how much of the original viewport's size should be covered by the 2D signed distance field. This SDF can be sampled in CanvasItem shaders and is also used for GPUParticles2D collision. Higher values allow portions of occluders located outside the viewport to still be taken into account in the generated signed distance field, at the cost of performance. If you notice particles falling through LightOccluder2Ds as the occluders leave the viewport, increase this setting.

The percentage is added on each axis and on both sides. For example, with the default SDF_OVERSIZE_120_PERCENT, the signed distance field will cover 20% of the viewport's size outside the viewport on each side (top, right, bottom, left).

SDFScale **sdf_scale** = `1` 🔗

- void **set_sdf_scale**(value: SDFScale)
- SDFScale **get_sdf_scale**()

The resolution scale to use for the 2D signed distance field. Higher values lead to a more precise and more stable signed distance field as the camera moves, at the cost of performance.

---

bool **snap_2d_transforms_to_pixel** = `false` 🔗

- void **set_snap_2d_transforms_to_pixel**(value: bool)
- bool **is_snap_2d_transforms_to_pixel_enabled**()

If `true`, CanvasItem nodes will internally snap to full pixels. Their position can still be sub-pixel, but the decimals will not have effect. This can lead to a crisper appearance at the cost of less smooth movement, especially when Camera2D smoothing is enabled.

---

bool **snap_2d_vertices_to_pixel** = `false` 🔗

- void **set_snap_2d_vertices_to_pixel**(value: bool)
- bool **is_snap_2d_vertices_to_pixel_enabled**()

If `true`, vertices of CanvasItem nodes will snap to full pixels. Only affects the final vertex positions, not the transforms. This can lead to a crisper appearance at the cost of less smooth movement, especially when Camera2D smoothing is enabled.

---

float **texture_mipmap_bias** = `0.0` 🔗

- void **set_texture_mipmap_bias**(value: float)
- float **get_texture_mipmap_bias**()

Affects the final texture sharpness by reading from a lower or higher mipmap (also called "texture LOD bias"). Negative values make mipmapped textures sharper but grainier when viewed at a distance, while positive values make mipmapped textures blurrier (even when up close).

Enabling temporal antialiasing (use_taa) will automatically apply a `-0.5` offset to this value, while enabling FXAA (screen_space_aa) will automatically apply a `-0.25` offset to this value. If both TAA and FXAA are enabled at the same time, an offset of `-0.75` is applied to this value.

Note: If scaling_3d_scale is lower than `1.0` (exclusive), texture_mipmap_bias is used to adjust the automatic mipmap bias which is calculated internally based on the scale factor. The formula for this is `log2(scaling_3d_scale) + mipmap_bias`.

To control this property on the root viewport, set the ProjectSettings.rendering/textures/default_filters/texture_mipmap_bias project setting.

---

bool **transparent_bg** = `false`

- void **set_transparent_background**(value: bool)
- bool **has_transparent_background**()

If `true`, the viewport should render its background as transparent.

---

bool **use_debanding** = `false`

- void **set_use_debanding**(value: bool)
- bool **is_using_debanding**()

If `true`, uses a fast post-processing filter to make banding significantly less visible in 3D. 2D rendering is *not* affected by debanding unless the Environment.background_mode is Environment.BG_CANVAS. See also ProjectSettings.rendering/anti_aliasing/quality/use_debanding.

In some cases, debanding may introduce a slightly noticeable dithering pattern. It's recommended to enable debanding only when actually needed since the dithering pattern will make lossless-compressed screenshots larger.

---

bool **use_hdr_2d** = `false`

- void **set_use_hdr_2d**(value: bool)
- bool **is_using_hdr_2d**()

If `true`, 2D rendering will use an high dynamic range (HDR) format framebuffer matching the bit depth of the 3D framebuffer. When using the Forward+ renderer this will be an `RGBA16` framebuffer, while when using the Mobile renderer it will be an `RGB10_A2` framebuffer.

Additionally, 2D rendering will take place in linear color space and will be converted to sRGB space immediately before blitting to the screen (if the Viewport is attached to the screen). Practically speaking, this means that the end result of the Viewport will not be clamped into the `0-1` range and can be used in 3D rendering without color space adjustments. This allows 2D rendering to take advantage of effects requiring high dynamic range (e.g. 2D glow) as well as substantially improves the appearance of effects requiring highly detailed gradients.

Note: This setting will have no effect when using the GL Compatibility renderer as the GL Compatibility renderer always renders in low dynamic range for performance reasons.

---

bool **use_occlusion_culling** = `false`

- void **set_use_occlusion_culling**(value: bool)
- bool **is_using_occlusion_culling**()

If `true`, OccluderInstance3D nodes will be usable for occlusion culling in 3D for this viewport. For the root viewport, ProjectSettings.rendering/occlusion_culling/use_occlusion_culling must be set to `true` instead.

Note: Enabling occlusion culling has a cost on the CPU. Only enable occlusion culling if you actually plan to use it, and think whether your scene can actually benefit from occlusion culling. Large, open scenes with few or no objects blocking the view will generally not benefit much from occlusion culling. Large open scenes generally benefit more from mesh LOD and visibility ranges (GeometryInstance3D.visibility_range_begin and GeometryInstance3D.visibility_range_end) compared to occlusion culling.

Note: Due to memory constraints, occlusion culling is not supported by default in Web export templates. It can be enabled by compiling custom Web export templates with `module_raycast_enabled=yes`.

---

bool **use_taa** = `false`

- void **set_use_taa**(value: bool)
- bool **is_using_taa**()

Enables Temporal Anti-Aliasing for this viewport. TAA works by jittering the camera and accumulating the images of the last rendered frames, motion vector rendering is used to account for camera and object motion.

Note: The implementation is not complete yet, some visual instances such as particles and skinned meshes may show artifacts.

---

bool **use_xr** = `false`

- void **set_use_xr**(value: bool)
- bool **is_using_xr**()

If `true`, the viewport will use the primary XR interface to render XR output. When applicable this can result in a stereoscopic image and the resulting render being output to a headset.

VRSMode **vrs_mode** = `0`

- void **set_vrs_mode**(value: VRSMode)
- VRSMode **get_vrs_mode**()

The Variable Rate Shading (VRS) mode that is used for this viewport. Note, if hardware does not support VRS this property is ignored.

Texture2D **vrs_texture**

- void **set_vrs_texture**(value: Texture2D)
- Texture2D **get_vrs_texture**()

Texture to use when vrs_mode is set to VRS_TEXTURE.

The texture *must* use a lossless compression format so that colors can be matched precisely. The following VRS densities are mapped to various colors, with brighter colors representing a lower level of shading precision:

```
- 1×1 = rgb(0, 0, 0)     - #000000
- 1×2 = rgb(0, 85, 0)    - #005500
- 2×1 = rgb(85, 0, 0)    - #550000
- 2×2 = rgb(85, 85, 0)   - #555500
- 2×4 = rgb(85, 170, 0)  - #55aa00
- 4×2 = rgb(170, 85, 0)  - #aa5500
- 4×4 = rgb(170, 170, 0) - #aaaa00
- 4×8 = rgb(170, 255, 0) - #aaff00 - Not supported on most hardware
- 8×4 = rgb(255, 170, 0) - #ffaa00 - Not supported on most hardware
- 8×8 = rgb(255, 255, 0) - #ffff00 - Not supported on most hardware
```

VRSUpdateMode **vrs_update_mode** = `1`

- void **set_vrs_update_mode**(value: VRSUpdateMode)
- VRSUpdateMode **get_vrs_update_mode**()

Sets the update mode for Variable Rate Shading (VRS) for the viewport. VRS requires the input texture to be converted to the format usable by the VRS method supported by the hardware. The update mode defines how often this happens. If the GPU does not support VRS, or VRS is not enabled, this property is ignored.

World2D **world_2d**

- void **set_world_2d**(value: World2D)
- World2D **get_world_2d**()

The custom World2D which can be used as 2D environment source.

World3D **world_3d**

- void **set_world_3d**(value: World3D)
- World3D **get_world_3d**()

The custom World3D which can be used as 3D environment source.

Method Descriptions

World2D **find_world_2d**() const

Returns the first valid World2D for this viewport, searching the world_2d property of itself and any Viewport ancestor.

World3D **find_world_3d**() const

Returns the first valid World3D for this viewport, searching the world_3d property of itself and any Viewport ancestor.

Camera2D **get_camera_2d**() const

Returns the currently active 2D camera. Returns null if there are no active cameras.

Camera3D **get_camera_3d**() const

Returns the currently active 3D camera.

bool **get_canvas_cull_mask_bit**(layer: int) const

Returns an individual bit on the rendering layer mask.

Array[Window] **get_embedded_subwindows**() const

Returns a list of the visible embedded Windows inside the viewport.

Note: Windows inside other viewports will not be listed.

Transform2D **get_final_transform**() const

Returns the transform from the viewport's coordinate system to the embedder's coordinate system.

Vector2 **get_mouse_position**() const

Returns the mouse's position in this **Viewport** using the coordinate system of this **Viewport**.

PositionalShadowAtlasQuadrantSubdiv **get_positional_shadow_atlas_quadrant_subdiv**(quadrant: int) const 🔗

Returns the positional shadow atlas quadrant subdivision of the specified quadrant.

---

int **get_render_info**(type: RenderInfoType, info: RenderInfo) 🔗

Returns rendering statistics of the given type. See RenderInfoType and RenderInfo for options.

Transform2D **get_screen_transform**() const 🔗

Returns the transform from the Viewport's coordinates to the screen coordinates of the containing window manager window.

ViewportTexture **get_texture**() const 🔗

Returns the viewport's texture.

Note: When trying to store the current texture (e.g. in a file), it might be completely black or outdated if used too early, especially when used in e.g. Node._ready. To make sure the texture you get is correct, you can await RenderingServer.frame_post_draw signal.

```
func _ready():
    await RenderingServer.frame_post_draw
    $Viewport.get_texture().get_image().save_png("user://Screenshot.png")
```

RID **get_viewport_rid**() const 🔗

Returns the viewport's RID from the RenderingServer.

Rect2 **get_visible_rect**() const 🔗

Returns the visible rectangle in global screen coordinates.

Variant **gui_get_drag_data**() const 🔗

Returns the drag data from the GUI, that was previously returned by Control._get_drag_data.

Control **gui_get_focus_owner**() const 🔗

Returns the Control having the focus within this viewport. If no Control has the focus, returns null.

Control **gui_get_hovered_control**() const

Returns the Control that the mouse is currently hovering over in this viewport. If no Control has the cursor, returns null.

Typically the leaf Control node or deepest level of the subtree which claims hover. This is very useful when used together with Node.is_ancestor_of to find if the mouse is within a control tree.

---

bool **gui_is_drag_successful**() const

Returns `true` if the drag operation is successful.

---

bool **gui_is_dragging**() const

Returns `true` if the viewport is currently performing a drag operation.

Alternative to Node.NOTIFICATION_DRAG_BEGIN and Node.NOTIFICATION_DRAG_END when you prefer polling the value.

---

void **gui_release_focus**()

Removes the focus from the currently focused Control within this viewport. If no Control has the focus, does nothing.

---

bool **is_input_handled**() const

Returns whether the current InputEvent has been handled. Input events are not handled until set_input_as_handled has been called during the lifetime of an InputEvent.

This is usually done as part of input handling methods like Node._input, Control._gui_input or others, as well as in corresponding signal handlers.

If handle_input_locally is set to `false`, this method will try finding the first parent viewport that is set to handle input locally, and return its value for is_input_handled instead.

---

void **push_input**(event: InputEvent, in_local_coords: bool = false)

Triggers the given `event` in this **Viewport**. This can be used to pass an InputEvent between viewports, or to locally apply inputs that were sent over the network or saved to a file.

If `in_local_coords` is `false`, the event's position is in the embedder's coordinates and will be converted to viewport coordinates. If `in_local_coords` is `true`, the event's position is in viewport coordinates.

While this method serves a similar purpose as Input.parse_input_event, it does not remap the specified `event` based on project settings like ProjectSettings.input_devices/pointing/emulate_touch_from_mouse.

Calling this method will propagate calls to child nodes for following methods in the given order:

- Node._input
- Control._gui_input for Control nodes
- Node._shortcut_input
- Node._unhandled_key_input
- Node._unhandled_input

If an earlier method marks the input as handled via set_input_as_handled, any later method in this list will not be called.

If none of the methods handle the event and physics_object_picking is `true`, the event is used for physics object picking.

---

void **push_text_input**(text: String)

Helper method which calls the `set_text()` method on the currently focused Control, provided that it is defined (e.g. if the focused Control is Button or LineEdit).

---

void **push_unhandled_input**(event: InputEvent, in_local_coords: bool = false)

Deprecated: Use push_input instead.

Triggers the given `event` in this **Viewport**. This can be used to pass an InputEvent between viewports, or to locally apply inputs that were sent over the network or saved to a file.

If `in_local_coords` is `false`, the event's position is in the embedder's coordinates and will be converted to viewport coordinates. If `in_local_coords` is `true`, the event's position is in viewport coordinates.

Calling this method will propagate calls to child nodes for following methods in the given order:

- Node._shortcut_input
- Node._unhandled_key_input
- Node._unhandled_input

If an earlier method marks the input as handled via set_input_as_handled, any later method in this list will not be called.

If none of the methods handle the event and physics_object_picking is `true`, the event is used for physics object picking.

Note: This method doesn't propagate input events to embedded Windows or SubViewports.

---

void **set_canvas_cull_mask_bit**(layer: int, enable: bool)

Set/clear individual bits on the rendering layer mask. This simplifies editing this **Viewport**'s layers.

---

void **set_input_as_handled**() 🔗

Stops the input from propagating further down the SceneTree.

Note: This does not affect the methods in Input, only the way events are propagated.

---

void **set_positional_shadow_atlas_quadrant_subdiv**(quadrant: int, subdiv: PositionalShadowAtlasQuadrantSubdiv) 🔗

Sets the number of subdivisions to use in the specified quadrant. A higher number of subdivisions allows you to have more shadows in the scene at once, but reduces the quality of the shadows. A good practice is to have quadrants with a varying number of subdivisions and to have as few subdivisions as possible.

---

void **update_mouse_cursor_state**() 🔗

Force instantly updating the display based on the current mouse cursor position. This includes updating the mouse cursor shape and sending necessary Control.mouse_entered, CollisionObject2D.mouse_entered, CollisionObject3D.mouse_entered and Window.mouse_entered signals and their respective `mouse_exited` counterparts.

---

void **warp_mouse**(position: Vector2) 🔗

Moves the mouse pointer to the specified position in this **Viewport** using the coordinate system of this **Viewport**.

Note: warp_mouse is only supported on Windows, macOS and Linux. It has no effect on Android, iOS and Web.

VisibleOnScreenEnabler2D

Inherits: VisibleOnScreenNotifier2D **<** Node2D **<** CanvasItem **<** Node **<** Object

A rectangular region of 2D space that, when visible on screen, enables a target node.

Description

VisibleOnScreenEnabler2D contains a rectangular region of 2D space and a target node. The target node will be automatically enabled (via its Node.process_mode property) when any part of this region becomes visible on the screen, and automatically disabled otherwise. This can for example be used to activate enemies only when the player approaches them.

See VisibleOnScreenNotifier2D if you only want to be notified when the region is visible on screen.

Note: VisibleOnScreenEnabler2D uses the render culling code to determine whether it's visible on screen, so it won't function unless CanvasItem.visible is set to `true`.

Properties

EnableMode	enable_mode	`0`
NodePath	enable_node_path	`NodePath("..")`

Enumerations

enum **EnableMode**:

EnableMode **ENABLE_MODE_INHERIT** = `0`

Corresponds to Node.PROCESS_MODE_INHERIT.

EnableMode **ENABLE_MODE_ALWAYS** = `1`

Corresponds to Node.PROCESS_MODE_ALWAYS.

EnableMode **ENABLE_MODE_WHEN_PAUSED** = `2`

Corresponds to Node.PROCESS_MODE_WHEN_PAUSED.

Property Descriptions

EnableMode **enable_mode** = `0`

- void **set_enable_mode**(value: EnableMode)
- EnableMode **get_enable_mode**()

Determines how the target node is enabled. Corresponds to ProcessMode. When the node is disabled, it always uses Node.PROCESS_MODE_DISABLED.

NodePath **enable_node_path** = `NodePath("..")`

- void **set_enable_node_path**(value: NodePath)
- NodePath **get_enable_node_path**()

The path to the target node, relative to the **VisibleOnScreenEnabler2D**. The target node is cached; it's only assigned when setting this property (if the **VisibleOnScreenEnabler2D** is inside the scene tree) and every time the **VisibleOnScreenEnabler2D** enters the scene tree. If the path is empty, no node will be affected. If the path is invalid, an error is also generated.

VisibleOnScreenEnabler3D

Inherits: VisibleOnScreenNotifier3D **<** VisualInstance3D **<** Node3D **<** Node **<** Object

A box-shaped region of 3D space that, when visible on screen, enables a target node.

Description

VisibleOnScreenEnabler3D contains a box-shaped region of 3D space and a target node. The target node will be automatically enabled (via its Node.process_mode property) when any part of this region becomes visible on the screen, and automatically disabled otherwise. This can for example be used to activate enemies only when the player approaches them.

See VisibleOnScreenNotifier3D if you only want to be notified when the region is visible on screen.

Note: VisibleOnScreenEnabler3D uses an approximate heuristic that doesn't take walls and other occlusion into account, unless occlusion culling is used. It also won't function unless Node3D.visible is set to `true`.

Properties

EnableMode	enable_mode	`0`
NodePath	enable_node_path	`NodePath("..")`

Enumerations

enum **EnableMode**:

EnableMode **ENABLE_MODE_INHERIT** = `0`

Corresponds to Node.PROCESS_MODE_INHERIT.

EnableMode **ENABLE_MODE_ALWAYS** = `1`

Corresponds to Node.PROCESS_MODE_ALWAYS.

EnableMode **ENABLE_MODE_WHEN_PAUSED** = `2`

Corresponds to Node.PROCESS_MODE_WHEN_PAUSED.

Property Descriptions

EnableMode **enable_mode** = `0`

- void **set_enable_mode**(value: EnableMode)
- EnableMode **get_enable_mode**()

Determines how the target node is enabled. Corresponds to ProcessMode. When the node is disabled, it always uses Node.PROCESS_MODE_DISABLED.

NodePath **enable_node_path** = `NodePath("..")`

- void **set_enable_node_path**(value: NodePath)
- NodePath **get_enable_node_path**()

The path to the target node, relative to the **VisibleOnScreenEnabler3D**. The target node is cached; it's only assigned when setting this property (if the **VisibleOnScreenEnabler3D** is inside the scene tree) and every time the **VisibleOnScreenEnabler3D** enters the scene tree. If the path is empty, no node will be affected. If the path is invalid, an error is also generated.

VisibleOnScreenNotifier2D

Inherits: Node2D **<** CanvasItem **<** Node **<** Object

Inherited By: VisibleOnScreenEnabler2D

A rectangular region of 2D space that detects whether it is visible on screen.

Description

VisibleOnScreenNotifier2D represents a rectangular region of 2D space. When any part of this region becomes visible on screen or in a viewport, it will emit a screen_entered signal, and likewise it will emit a screen_exited signal when no part of it remains visible.

If you want a node to be enabled automatically when this region is visible on screen, use VisibleOnScreenEnabler2D.

Note: VisibleOnScreenNotifier2D uses the render culling code to determine whether it's visible on screen, so it won't function unless CanvasItem.visible is set to `true`.

Tutorials

- 2D Dodge The Creeps Demo [https://godotengine.org/asset-library/asset/2712]

Properties

Rect2	rect	`Rect2(-10, -10, 20, 20)`

Methods

bool	is_on_screen() const

Signals

screen_entered()

Emitted when the VisibleOnScreenNotifier2D enters the screen.

---

screen_exited()

Emitted when the VisibleOnScreenNotifier2D exits the screen.

Property Descriptions

Rect2 **rect** = `Rect2(-10, -10, 20, 20)`

- void **set_rect**(value: Rect2)
- Rect2 **get_rect**()

The VisibleOnScreenNotifier2D's bounding rectangle.

Method Descriptions

bool **is_on_screen**() const

If `true`, the bounding rectangle is on the screen.

Note: It takes one frame for the **VisibleOnScreenNotifier2D**'s visibility to be determined once added to the scene tree, so this method will always return `false` right after it is instantiated, before the draw pass.

VisibleOnScreenNotifier3D

Inherits: VisualInstance3D **<** Node3D **<** Node **<** Object

Inherited By: VisibleOnScreenEnabler3D

A box-shaped region of 3D space that detects whether it is visible on screen.

Description

VisibleOnScreenNotifier3D represents a box-shaped region of 3D space. When any part of this region becomes visible on screen or in a Camera3D's view, it will emit a screen_entered signal, and likewise it will emit a screen_exited signal when no part of it remains visible.

If you want a node to be enabled automatically when this region is visible on screen, use VisibleOnScreenEnabler3D.

Note: VisibleOnScreenNotifier3D uses an approximate heuristic that doesn't take walls and other occlusion into account, unless occlusion culling is used. It also won't function unless Node3D.visible is set to `true`.

Properties

AABB	aabb	`AABB(-1, -1, -1, 2, 2, 2)`

Methods

bool	is_on_screen() const

Signals

screen_entered()

Emitted when the **VisibleOnScreenNotifier3D** enters the screen.

screen_exited()

Emitted when the **VisibleOnScreenNotifier3D** exits the screen.

Property Descriptions

AABB **aabb** = `AABB(-1, -1, -1, 2, 2, 2)`

- void **set_aabb**(value: AABB)
- AABB **get_aabb**()

The **VisibleOnScreenNotifier3D**'s bounding box.

Method Descriptions

bool **is_on_screen**() const

Returns `true` if the bounding box is on the screen.

Note: It takes one frame for the **VisibleOnScreenNotifier3D**'s visibility to be assessed once added to the scene tree, so this method will always return `false` right after it is instantiated.

VisualInstance3D

Inherits: Node3D **<** Node **<** Object

Inherited By: Decal, FogVolume, GeometryInstance3D, GPUParticlesAttractor3D, GPUParticlesCollision3D, Light3D, LightmapGI, OccluderInstance3D, ReflectionProbe, RootMotionView, VisibleOnScreenNotifier3D, VoxelGI

Parent of all visual 3D nodes.

Description

The **VisualInstance3D** is used to connect a resource to a visual representation. All visual 3D nodes inherit from the **VisualInstance3D**. In general, you should not access the **VisualInstance3D** properties directly as they are accessed and managed by the nodes that inherit from **VisualInstance3D**. **VisualInstance3D** is the node representation of the RenderingServer instance.

Properties

int	layers	`1`
float	sorting_offset	`0.0`
bool	sorting_use_aabb_center	

Methods

AABB	_get_aabb() virtual const
AABB	get_aabb() const
RID	get_base() const
RID	get_instance() const
bool	get_layer_mask_value(layer_number: int) const
void	set_base(base: RID)
void	set_layer_mask_value(layer_number: int, value: bool)

Property Descriptions

int **layers** = `1` 🔗

- void **set_layer_mask**(value: int)
- int **get_layer_mask**()

The render layer(s) this **VisualInstance3D** is drawn on.

This object will only be visible for Camera3Ds whose cull mask includes any of the render layers this **VisualInstance3D** is set to.

For Light3Ds, this can be used to control which **VisualInstance3D**s are affected by a specific light. For GPUParticles3D, this can be used to control which particles are effected by a specific attractor. For Decals, this can be used to control which **VisualInstance3D**s are affected by a specific decal.

To adjust layers more easily using a script, use get_layer_mask_value and set_layer_mask_value.

Note: VoxelGI, SDFGI and LightmapGI will always take all layers into account to determine what contributes to global illumination. If this is an issue, set GeometryInstance3D.gi_mode to GeometryInstance3D.GI_MODE_DISABLED for meshes and Light3D.light_bake_mode to Light3D.BAKE_DISABLED for lights to exclude them from global illumination.

---

float **sorting_offset** = `0.0` 🔗

- void **set_sorting_offset**(value: float)
- float **get_sorting_offset**()

The amount by which the depth of this **VisualInstance3D** will be adjusted when sorting by depth. Uses the same units as the engine (which are typically meters). Adjusting it to a higher value will make the **VisualInstance3D** reliably draw on top of other **VisualInstance3D**s that are otherwise positioned at the same spot. To ensure it always draws on top of other objects around it (not positioned at the same spot), set the value to be greater than the distance between this **VisualInstance3D** and the other nearby **VisualInstance3D**s.

---

bool **sorting_use_aabb_center** 🔗

- void **set_sorting_use_aabb_center**(value: bool)

- bool **is_sorting_use_aabb_center**()

If `true`, the object is sorted based on the AABB center. The object will be sorted based on the global position otherwise.

The AABB center based sorting is generally more accurate for 3D models. The position based sorting instead allows to better control the drawing order when working with GPUParticles3D and CPUParticles3D.

Method Descriptions

AABB **_get_aabb**() virtual const

There is currently no description for this method. Please help us by contributing one!

AABB **get_aabb**() const

Returns the AABB (also known as the bounding box) for this **VisualInstance3D**.

RID **get_base**() const

Returns the RID of the resource associated with this **VisualInstance3D**. For example, if the Node is a MeshInstance3D, this will return the RID of the associated Mesh.

RID **get_instance**() const

Returns the RID of this instance. This RID is the same as the RID returned by RenderingServer.instance_create. This RID is needed if you want to call RenderingServer functions directly on this **VisualInstance3D**.

---

bool **get_layer_mask_value**(layer_number: int) const 🔗

Returns whether or not the specified layer of the layers is enabled, given a `layer_number` between 1 and 20.

---

void **set_base**(base: RID) 🔗

Sets the resource that is instantiated by this **VisualInstance3D**, which changes how the engine handles the **VisualInstance3D** under the hood. Equivalent to RenderingServer.instance_set_base.

---

void **set_layer_mask_value**(layer_number: int, value: bool) 🔗

Based on `value`, enables or disables the specified layer in the layers, given a `layer_number` between 1 and 20.

VoxelGI

Inherits: VisualInstance3D **<** Node3D **<** Node **<** Object

Real-time global illumination (GI) probe.

Description

VoxelGIs are used to provide high-quality real-time indirect light and reflections to scenes. They precompute the effect of objects that emit light and the effect of static geometry to simulate the behavior of complex light in real-time. **VoxelGI**s need to be baked before having a visible effect. However, once baked, dynamic objects will receive light from them. Furthermore, lights can be fully dynamic or baked.

Note: VoxelGI is only supported in the Forward+ rendering method, not Mobile or Compatibility.

Procedural generation: VoxelGI can be baked in an exported project, which makes it suitable for procedurally generated or user-built levels as long as all the geometry is generated in advance. For games where geometry is generated at any time during gameplay, SDFGI is more suitable (see Environment.sdfgi_enabled).

Performance: VoxelGI is relatively demanding on the GPU and is not suited to low-end hardware such as integrated graphics (consider LightmapGI instead). To improve performance, adjust ProjectSettings.rendering/global_illumination/voxel_gi/quality and enable ProjectSettings.rendering/global_illumination/gi/use_half_resolution in the Project Settings. To provide a fallback for low-end hardware, consider adding an option to disable **VoxelGI** in your project's options menus. A **VoxelGI** node can be disabled by hiding it.

Note: Meshes should have sufficiently thick walls to avoid light leaks (avoid one-sided walls). For interior levels, enclose your level geometry in a sufficiently large box and bridge the loops to close the mesh. To further prevent light leaks, you can also strategically place temporary MeshInstance3D nodes with their GeometryInstance3D.gi_mode set to GeometryInstance3D.GI_MODE_STATIC. These temporary nodes can then be hidden after baking the **VoxelGI** node.

Tutorials

- Using Voxel global illumination
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

CameraAttributes	camera_attributes	
VoxelGIData	data	
Vector3	size	`Vector3(20, 20, 20)`
Subdiv	subdiv	`1`

Methods

void	bake(from_node: Node = null, create_visual_debug: bool = false)
void	debug_bake()

Enumerations

enum **Subdiv**: 🔗

Subdiv **SUBDIV_64** = `0`

Use 64 subdivisions. This is the lowest quality setting, but the fastest. Use it if you can, but especially use it on lower-end hardware.

Subdiv **SUBDIV_128** = `1`

Use 128 subdivisions. This is the default quality setting.

Subdiv **SUBDIV_256** = `2`

Use 256 subdivisions.

Subdiv **SUBDIV_512** = `3`

Use 512 subdivisions. This is the highest quality setting, but the slowest. On lower-end hardware, this could cause the GPU to stall.

Subdiv **SUBDIV_MAX** = `4`

Represents the size of the Subdiv enum.

Property Descriptions

CameraAttributes **camera_attributes** 🔗

- void **set_camera_attributes**(value: CameraAttributes)
- CameraAttributes **get_camera_attributes**()

The CameraAttributes resource that specifies exposure levels to bake at. Auto-exposure and non exposure properties will be ignored. Exposure settings should be used to reduce the dynamic range present when baking. If exposure is too high, the **VoxelGI** will have banding artifacts or may have over-exposure artifacts.

VoxelGIData **data** 🔗

- void **set_probe_data**(value: VoxelGIData)
- VoxelGIData **get_probe_data**()

The VoxelGIData resource that holds the data for this **VoxelGI**.

Vector3 **size** = `Vector3(20, 20, 20)` 🔗

- void **set_size**(value: Vector3)
- Vector3 **get_size**()

The size of the area covered by the **VoxelGI**. If you make the size larger without increasing the subdivisions with subdiv, the size of each cell will increase and result in lower detailed lighting.

Note: Size is clamped to 1.0 unit or more on each axis.

Subdiv **subdiv** = `1` 🔗

- void **set_subdiv**(value: Subdiv)
- Subdiv **get_subdiv**()

Number of times to subdivide the grid that the **VoxelGI** operates on. A higher number results in finer detail and thus higher visual quality, while lower numbers result in better performance.

Method Descriptions

void **bake**(from_node: Node = null, create_visual_debug: bool = false) 🔗

Bakes the effect from all GeometryInstance3Ds marked with GeometryInstance3D.GI_MODE_STATIC and Light3Ds marked with either Light3D.BAKE_STATIC or Light3D.BAKE_DYNAMIC. If `create_visual_debug` is `true`, after baking the light, this will generate a MultiMesh that has a cube representing each solid cell with each cube colored to the cell's albedo color. This can be used to visualize the **VoxelGI**'s data and debug any issues that may be occurring.

Note: bake works from the editor and in exported projects. This makes it suitable for procedurally generated or user-built levels. Baking a **VoxelGI** node generally takes from 5 to 20 seconds in most scenes. Reducing subdiv can speed up baking.

Note: GeometryInstance3Ds and Light3Ds must be fully ready before bake is called. If you are procedurally creating those and some meshes or lights are missing from your baked **VoxelGI**, use `call_deferred("bake")` instead of calling bake directly.

---

void **debug_bake**() 🔗

Calls bake with `create_visual_debug` enabled.

VScrollBar

Inherits: ScrollBar **<** Range **<** Control **<** CanvasItem **<** Node **<** Object

A vertical scrollbar that goes from top (min) to bottom (max).

Description

A vertical scrollbar, typically used to navigate through content that extends beyond the visible height of a control. It is a Range-based control and goes from top (min) to bottom (max). Note that this direction is the opposite of VSlider's.

Properties

BitField[SizeFlags]	size_flags_horizontal	`0` (overrides Control)
BitField[SizeFlags]	size_flags_vertical	`1` (overrides Control)

VSeparator

Inherits: Separator **<** Control **<** CanvasItem **<** Node **<** Object

A vertical line used for separating other controls.

Description

A vertical separator used for separating other controls that are arranged **horizontally**. **VSeparator** is purely visual and normally drawn as a StyleBoxLine.

VSlider

Inherits: Slider **<** Range **<** Control **<** CanvasItem **<** Node **<** Object

A vertical slider that goes from bottom (min) to top (max).

Description

A vertical slider, used to adjust a value by moving a grabber along a vertical axis. It is a Range-based control and goes from bottom (min) to top (max). Note that this direction is the opposite of VScrollBar's.

Properties

BitField[SizeFlags]	size_flags_horizontal	`0`	(overrides Control)
BitField[SizeFlags]	size_flags_vertical	`1`	(overrides Control)

VSplitContainer

Inherits: SplitContainer **<** Container **<** Control **<** CanvasItem **<** Node **<** Object

A container that splits two child controls vertically and provides a grabber for adjusting the split ratio.

Description

A container that accepts only two child controls, then arranges them vertically and creates a divisor between them. The divisor can be dragged around to change the size relation between the child controls.

Tutorials

- Using Containers

Window

Inherits: Viewport **<** Node **<** Object

Inherited By: AcceptDialog, Popup

Base class for all windows, dialogs, and popups.

Description

A node that creates a window. The window can either be a native system window or embedded inside another **Window** (see Viewport.gui_embed_subwindows).

At runtime, **Window**s will not close automatically when requested. You need to handle it manually using the close_requested signal (this applies both to pressing the close button and clicking outside of a popup).

Properties

bool	always_on_top	`false`
bool	auto_translate	`true`
bool	borderless	`false`
ContentScaleAspect	content_scale_aspect	`0`
float	content_scale_factor	`1.0`

ContentScaleMode	content_scale_mode	`0`
Vector2i	content_scale_size	`Vector2i(0, 0)`
ContentScaleStretch	content_scale_stretch	`0`
int	current_screen	
bool	exclusive	`false`
bool	extend_to_title	`false`
bool	force_native	`false`
WindowInitialPosition	initial_position	`0`
bool	keep_title_visible	`false`
Vector2i	max_size	`Vector2i(0, 0)`
Vector2i	min_size	`Vector2i(0, 0)`
Mode	mode	`0`
bool	mouse_passthrough	`false`
PackedVector2Array	mouse_passthrough_polygon	`PackedVector2Array()`

bool	popup_window	`false`
Vector2i	position	`Vector2i(0, 0)`
Vector2i	size	`Vector2i(100, 100)`
Theme	theme	
StringName	theme_type_variation	`&""`
String	title	`""`
bool	transient	`false`
bool	transient_to_focused	`false`
bool	transparent	`false`
bool	unfocusable	`false`
bool	unresizable	`false`
bool	visible	`true`
bool	wrap_controls	`false`

Methods

Vector2	_get_contents_minimum_size() virtual const
void	add_theme_color_override(name: StringName, color: Color)
void	add_theme_constant_override(name: StringName, constant: int)
void	add_theme_font_override(name: StringName, font: Font)
void	add_theme_font_size_override(name: StringName, font_size: int)
void	add_theme_icon_override(name: StringName, texture: Texture2D)
void	add_theme_stylebox_override(name: StringName, stylebox: StyleBox)
void	begin_bulk_theme_override()
bool	can_draw() const
void	child_controls_changed()
void	end_bulk_theme_override()

Vector2	get_contents_minimum_size() const
bool	get_flag(flag: Flags) const
LayoutDirection	get_layout_direction() const
Vector2i	get_position_with_decorations() const
Vector2i	get_size_with_decorations() const
Color	get_theme_color(name: StringName, theme_type: StringName = &"") const
int	get_theme_constant(name: StringName, theme_type: StringName = &"") const
float	get_theme_default_base_scale() const
Font	get_theme_default_font() const
int	get_theme_default_font_size() const
Font	get_theme_font(name: StringName, theme_type: StringName = &"") const
int	get_theme_font_size(name: StringName, theme_type: StringName = &"") const

Texture2D	get_theme_icon(name: StringName, theme_type: StringName = &"") const
StyleBox	get_theme_stylebox(name: StringName, theme_type: StringName = &"") const
int	get_window_id() const
void	grab_focus()
bool	has_focus() const
bool	has_theme_color(name: StringName, theme_type: StringName = &"") const
bool	has_theme_color_override(name: StringName) const
bool	has_theme_constant(name: StringName, theme_type: StringName = &"") const
bool	has_theme_constant_override(name: StringName) const
bool	has_theme_font(name: StringName, theme_type: StringName = &"") const
bool	has_theme_font_override(name: StringName) const
bool	has_theme_font_size(name: StringName, theme_type: StringName = &"") const

bool	has_theme_font_size_override(name: StringName) const
bool	has_theme_icon(name: StringName, theme_type: StringName = &"") const
bool	has_theme_icon_override(name: StringName) const
bool	has_theme_stylebox(name: StringName, theme_type: StringName = &"") const
bool	has_theme_stylebox_override(name: StringName) const
void	hide()
bool	is_embedded() const
bool	is_layout_rtl() const
bool	is_maximize_allowed() const
bool	is_using_font_oversampling() const
void	move_to_center()
void	move_to_foreground()
void	popup(rect: Rect2i = Rect2i(0, 0, 0, 0))

void	popup_centered(minsize: Vector2i = Vector2i(0, 0))
void	popup_centered_clamped(minsize: Vector2i = Vector2i(0, 0), fallback_ratio: float = 0.75)
void	popup_centered_ratio(ratio: float = 0.8)
void	popup_exclusive(from_node: Node, rect: Rect2i = Rect2i(0, 0, 0, 0))
void	popup_exclusive_centered(from_node: Node, minsize: Vector2i = Vector2i(0, 0))
void	popup_exclusive_centered_clamped(from_node: Node, minsize: Vector2i = Vector2i(0, 0), fallback_ratio: float = 0.75)
void	popup_exclusive_centered_ratio(from_node: Node, ratio: float = 0.8)
void	popup_exclusive_on_parent(from_node: Node, parent_rect: Rect2i)
void	popup_on_parent(parent_rect: Rect2i)
void	remove_theme_color_override(name: StringName)
void	remove_theme_constant_override(name: StringName)

void	remove_theme_font_override(name: StringName)
void	remove_theme_font_size_override(name: StringName)
void	remove_theme_icon_override(name: StringName)
void	remove_theme_stylebox_override(name: StringName)
void	request_attention()
void	reset_size()
void	set_flag(flag: Flags, enabled: bool)
void	set_ime_active(active: bool)
void	set_ime_position(position: Vector2i)
void	set_layout_direction(direction: LayoutDirection)
void	set_unparent_when_invisible(unparent: bool)
void	set_use_font_oversampling(enable: bool)
void	show()

Theme Properties

Color	title_color	`Color(0.875, 0.875, 0.875, 1)`
Color	title_outline_modulate	`Color(0, 0, 0, 1)`
int	close_h_offset	`18`
int	close_v_offset	`24`
int	resize_margin	`4`
int	title_height	`36`
int	title_outline_size	`0`
Font	title_font	
int	title_font_size	
Texture2D	close	
Texture2D	close_pressed	
StyleBox	embedded_border	
StyleBox	embedded_unfocused_border	

Signals

about_to_popup()

Emitted right after popup call, before the **Window** appears or does anything.

---

close_requested()

Emitted when the **Window**'s close button is pressed or when popup_window is enabled and user clicks outside the window.

This signal can be used to handle window closing, e.g. by connecting it to hide.

---

dpi_changed()

Emitted when the **Window**'s DPI changes as a result of OS-level changes (e.g. moving the window from a Retina display to a lower resolution one).

Note: Only implemented on macOS.

---

files_dropped(files: PackedStringArray)

Emitted when files are dragged from the OS file manager and dropped in the game window. The argument is a list of file paths.

Note that this method only works with native windows, i.e. the main window and **Window**-derived nodes when Viewport.gui_embed_subwindows is disabled in the main viewport.

Example usage:

```
func _ready():
    get_viewport().files_dropped.connect(on_files_dropped)

func on_files_dropped(files):
    print(files)
```

---

focus_entered()

Emitted when the **Window** gains focus.

---

focus_exited()

Emitted when the **Window** loses its focus.

---

go_back_requested()

Emitted when a go back request is sent (e.g. pressing the "Back" button on Android), right after Node.NOTIFICATION_WM_GO_BACK_REQUEST.

---

mouse_entered()

Emitted when the mouse cursor enters the **Window**'s visible area, that is not occluded behind other Controls or windows, provided its Viewport.gui_disable_input is `false` and regardless if it's currently focused or not.

---

mouse_exited()

Emitted when the mouse cursor leaves the **Window**'s visible area, that is not occluded behind other Controls or windows, provided its Viewport.gui_disable_input is `false` and regardless if it's currently focused or not.

---

theme_changed()

Emitted when the NOTIFICATION_THEME_CHANGED notification is sent.

---

titlebar_changed()

Emitted when window title bar decorations are changed, e.g. macOS window enter/exit full screen mode, or extend-to-title flag is changed.

visibility_changed() 🔗

Emitted when **Window** is made visible or disappears.

window_input(event: InputEvent) 🔗

Emitted when the **Window** is currently focused and receives any input, passing the received event as an argument. The event's position, if present, is in the embedder's coordinate system.

Enumerations

enum **Mode**: 🔗

Mode **MODE_WINDOWED** = `0`

Windowed mode, i.e. **Window** doesn't occupy the whole screen (unless set to the size of the screen).

Mode **MODE_MINIMIZED** = `1`

Minimized window mode, i.e. **Window** is not visible and available on window manager's window list. Normally happens when the minimize button is pressed.

Mode **MODE_MAXIMIZED** = `2`

Maximized window mode, i.e. **Window** will occupy whole screen area except task bar and still display its borders. Normally happens when the maximize button is pressed.

Mode **MODE_FULLSCREEN** = `3`

Full screen mode with full multi-window support.

Full screen window covers the entire display area of a screen and has no decorations. The display's video mode is not changed.

On Windows: Multi-window full-screen mode has a 1px border of the ProjectSettings.rendering/environment/defaults/default_clear_color color.

On macOS: A new desktop is used to display the running project.

Note: Regardless of the platform, enabling full screen will change the window size to match the monitor's size. Therefore, make sure your project supports multiple resolutions when enabling full screen mode.

Mode **MODE_EXCLUSIVE_FULLSCREEN** = `4`

A single window full screen mode. This mode has less overhead, but only one window can be open on a given screen at a time (opening a child window or application switching will trigger a full screen transition).

Full screen window covers the entire display area of a screen and has no border or decorations. The display's video mode is not changed.

On Windows: Depending on video driver, full screen transition might cause screens to go black for a moment.

On macOS: A new desktop is used to display the running project. Exclusive full screen mode prevents Dock and Menu from showing up when the mouse pointer is hovering the edge of the screen.

On Linux (X11): Exclusive full screen mode bypasses compositor.

Note: Regardless of the platform, enabling full screen will change the window size to match the monitor's size. Therefore, make sure your project supports multiple resolutions when enabling full screen mode.

---

enum **Flags**:

Flags **FLAG_RESIZE_DISABLED** = `0`

The window can't be resized by dragging its resize grip. It's still possible to resize the window using size. This flag is ignored for full screen windows.

Set with unresizable.

Flags **FLAG_BORDERLESS** = `1`

The window do not have native title bar and other decorations. This flag is ignored for full-screen windows. Set with borderless.

Flags **FLAG_ALWAYS_ON_TOP** = `2`

The window is floating on top of all other windows. This flag is ignored for full-screen windows. Set with always_on_top.

Flags **FLAG_TRANSPARENT** = `3`

The window background can be transparent. Set with transparent.

Note: This flag has no effect if either ProjectSettings.display/window/per_pixel_transparency/allowed, or the window's Viewport.transparent_bg is set to `false`.

Flags **FLAG_NO_FOCUS** = `4`

The window can't be focused. No-focus window will ignore all input, except mouse clicks. Set with unfocusable.

Flags **FLAG_POPUP** = `5`

Window is part of menu or OptionButton dropdown. This flag can't be changed when the window is visible. An active popup window will exclusively receive all input, without stealing focus from its parent. Popup windows are automatically closed when uses click outside it, or when an application is switched. Popup window must have transient parent set (see transient).

Note: This flag has no effect in embedded windows (unless said window is a Popup).

Flags **FLAG_EXTEND_TO_TITLE** = `6`

Window content is expanded to the full size of the window. Unlike borderless window, the frame is left intact and can be used to resize the window, title bar is transparent, but have minimize/maximize/close buttons. Set with extend_to_title.

Note: This flag is implemented only on macOS.

Note: This flag has no effect in embedded windows.

Flags **FLAG_MOUSE_PASSTHROUGH** = `7`

All mouse events are passed to the underlying window of the same application.

Note: This flag has no effect in embedded windows.

Flags **FLAG_MAX** = `8`

Max value of the Flags.

---

enum **ContentScaleMode**: 🔗

ContentScaleMode **CONTENT_SCALE_MODE_DISABLED** = `0`

The content will not be scaled to match the **Window**'s size.

ContentScaleMode **CONTENT_SCALE_MODE_CANVAS_ITEMS** = `1`

The content will be rendered at the target size. This is more performance-expensive than CONTENT_SCALE_MODE_VIEWPORT, but provides better results.

ContentScaleMode **CONTENT_SCALE_MODE_VIEWPORT** = `2`

The content will be rendered at the base size and then scaled to the target size. More performant than CONTENT_SCALE_MODE_CANVAS_ITEMS, but results in pixelated image.

---

enum **ContentScaleAspect**: 🔗

ContentScaleAspect **CONTENT_SCALE_ASPECT_IGNORE** = `0`

The aspect will be ignored. Scaling will simply stretch the content to fit the target size.

ContentScaleAspect **CONTENT_SCALE_ASPECT_KEEP** = `1`

The content's aspect will be preserved. If the target size has different aspect from the base one, the image will be centered and black bars will appear on left and right sides.

ContentScaleAspect **CONTENT_SCALE_ASPECT_KEEP_WIDTH** = `2`

The content can be expanded vertically. Scaling horizontally will result in keeping the width ratio and then black bars on left and right sides.

ContentScaleAspect **CONTENT_SCALE_ASPECT_KEEP_HEIGHT** = `3`

The content can be expanded horizontally. Scaling vertically will result in keeping the height ratio and then black bars on top and bottom sides.

ContentScaleAspect **CONTENT_SCALE_ASPECT_EXPAND** = `4`

The content's aspect will be preserved. If the target size has different aspect from the base one, the content will stay in the top-left corner and add an extra visible area in the stretched space.

---

enum **ContentScaleStretch**: 🔗

ContentScaleStretch **CONTENT_SCALE_STRETCH_FRACTIONAL** = `0`

The content will be stretched according to a fractional factor. This fills all the space available in the window, but allows "pixel wobble" to occur due to uneven pixel scaling.

ContentScaleStretch **CONTENT_SCALE_STRETCH_INTEGER** = `1`

The content will be stretched only according to an integer factor, preserving sharp pixels. This may leave a black background visible on the window's edges depending on the window size.

---

enum **LayoutDirection**: 🔗

LayoutDirection **LAYOUT_DIRECTION_INHERITED** = `0`

Automatic layout direction, determined from the parent window layout direction.

LayoutDirection **LAYOUT_DIRECTION_LOCALE** = `1`

Automatic layout direction, determined from the current locale.

LayoutDirection **LAYOUT_DIRECTION_LTR** = `2`

Left-to-right layout direction.

LayoutDirection **LAYOUT_DIRECTION_RTL** = `3`

Right-to-left layout direction.

---

enum **WindowInitialPosition**: 🔗

WindowInitialPosition **WINDOW_INITIAL_POSITION_ABSOLUTE** = `0`

Initial window position is determined by position.

WindowInitialPosition **WINDOW_INITIAL_POSITION_CENTER_PRIMARY_SCREEN** = `1`

Initial window position is the center of the primary screen.

WindowInitialPosition **WINDOW_INITIAL_POSITION_CENTER_MAIN_WINDOW_SCREEN** = `2`

Initial window position is the center of the main window screen.

WindowInitialPosition **WINDOW_INITIAL_POSITION_CENTER_OTHER_SCREEN** = `3`

Initial window position is the center of current_screen screen.

WindowInitialPosition **WINDOW_INITIAL_POSITION_CENTER_SCREEN_WITH_MOUSE_FOCUS** = `4`

Initial window position is the center of the screen containing the mouse pointer.

WindowInitialPosition **WINDOW_INITIAL_POSITION_CENTER_SCREEN_WITH_KEYBOARD_FOCUS** = `5`

Initial window position is the center of the screen containing the window with the keyboard focus.

Constants

NOTIFICATION_VISIBILITY_CHANGED = `30` 🔗

Emitted when **Window**'s visibility changes, right before visibility_changed.

NOTIFICATION_THEME_CHANGED = `32` 🔗

Sent when the node needs to refresh its theme items. This happens in one of the following cases:

- The theme property is changed on this node or any of its ancestors.
- The theme_type_variation property is changed on this node.
- The node enters the scene tree.

Note: As an optimization, this notification won't be sent from changes that occur while this node is outside of the scene tree. Instead, all of the theme item updates can be applied at once when the node enters the scene tree.

Property Descriptions

bool **always_on_top** = `false` 🔗

- void **set_flag**(flag: Flags, enabled: bool)
- bool **get_flag**(flag: Flags) const

If `true`, the window will be on top of all other windows. Does not work if transient is enabled.

---

bool **auto_translate** = `true` 🔗

- void **set_auto_translate**(value: bool)
- bool **is_auto_translating**()

Deprecated: Use Node.auto_translate_mode instead.

Toggles if any text should automatically change to its translated version depending on the current locale.

---

bool **borderless** = `false` 🔗

- void **set_flag**(flag: Flags, enabled: bool)
- bool **get_flag**(flag: Flags) const

If `true`, the window will have no borders.

ContentScaleAspect **content_scale_aspect** = `0` 🔗

- void **set_content_scale_aspect**(value: ContentScaleAspect)
- ContentScaleAspect **get_content_scale_aspect**()

Specifies how the content's aspect behaves when the **Window** is resized. The base aspect is determined by content_scale_size.

---

float **content_scale_factor** = `1.0` 🔗

- void **set_content_scale_factor**(value: float)
- float **get_content_scale_factor**()

Specifies the base scale of **Window**'s content when its size is equal to content_scale_size.

ContentScaleMode **content_scale_mode** = `0`

- void **set_content_scale_mode**(value: ContentScaleMode)
- ContentScaleMode **get_content_scale_mode**()

Specifies how the content is scaled when the **Window** is resized.

Vector2i **content_scale_size** = `Vector2i(0, 0)`

- void **set_content_scale_size**(value: Vector2i)
- Vector2i **get_content_scale_size**()

Base size of the content (i.e. nodes that are drawn inside the window). If non-zero, **Window**'s content will be scaled when the window is resized to a different size.

ContentScaleStretch **content_scale_stretch** = `0`

- void **set_content_scale_stretch**(value: ContentScaleStretch)
- ContentScaleStretch **get_content_scale_stretch**()

The policy to use to determine the final scale factor for 2D elements. This affects how content_scale_factor is applied, in addition to the automatic scale factor determined by content_scale_size.

---

int **current_screen**

- void **set_current_screen**(value: int)
- int **get_current_screen**()

The screen the window is currently on.

---

bool **exclusive** = `false`

- void **set_exclusive**(value: bool)
- bool **is_exclusive**()

If `true`, the **Window** will be in exclusive mode. Exclusive windows are always on top of their parent and will block all input going to the parent **Window**.

Needs transient enabled to work.

---

bool **extend_to_title** = `false` 🔗

- void **set_flag**(flag: Flags, enabled: bool)
- bool **get_flag**(flag: Flags) const

If `true`, the **Window** contents is expanded to the full size of the window, window title bar is transparent.

Note: This property is implemented only on macOS.

Note: This property only works with native windows.

---

bool **force_native** = `false` 🔗

- void **set_force_native**(value: bool)
- bool **get_force_native**()

If `true`, native window will be used regardless of parent viewport and project settings.

WindowInitialPosition **initial_position** = `0` 🔗

- void **set_initial_position**(value: WindowInitialPosition)
- WindowInitialPosition **get_initial_position**()

Specifies the initial type of position for the **Window**. See WindowInitialPosition constants.

---

bool **keep_title_visible** = `false`

- void **set_keep_title_visible**(value: bool)
- bool **get_keep_title_visible**()

If `true`, the **Window** width is expanded to keep the title bar text fully visible.

Vector2i **max_size** = `Vector2i(0, 0)`

- void **set_max_size**(value: Vector2i)
- Vector2i **get_max_size**()

If non-zero, the **Window** can't be resized to be bigger than this size.

Note: This property will be ignored if the value is lower than min_size.

Vector2i **min_size** = `Vector2i(0, 0)`

- void **set_min_size**(value: Vector2i)
- Vector2i **get_min_size**()

If non-zero, the **Window** can't be resized to be smaller than this size.

Note: This property will be ignored in favor of get_contents_minimum_size if wrap_controls is enabled and if its size is bigger.

Mode **mode** = `0`

- void **set_mode**(value: Mode)
- Mode **get_mode**()

Set's the window's current mode.

Note: Fullscreen mode is not exclusive full screen on Windows and Linux.

Note: This method only works with native windows, i.e. the main window and **Window**-derived nodes when Viewport.gui_embed_subwindows is disabled in the main viewport.

bool **mouse_passthrough** = `false`

- void **set_flag**(flag: Flags, enabled: bool)
- bool **get_flag**(flag: Flags) const

If `true`, all mouse events will be passed to the underlying window of the same application. See also mouse_passthrough_polygon.

Note: This property is implemented on Linux (X11), macOS and Windows.

Note: This property only works with native windows.

PackedVector2Array **mouse_passthrough_polygon** = `PackedVector2Array()`

- void **set_mouse_passthrough_polygon**(value: PackedVector2Array)
- PackedVector2Array **get_mouse_passthrough_polygon**()

Sets a polygonal region of the window which accepts mouse events. Mouse events outside the region will be passed through.

Passing an empty array will disable passthrough support (all mouse events will be intercepted by the window, which is the default behavior).

GDScript

```
# Set region, using Path2D node.
$Window.mouse_passthrough_polygon = $Path2D.curve.get_baked_points()

# Set region, using Polygon2D node.
$Window.mouse_passthrough_polygon = $Polygon2D.polygon

# Reset region to default.
$Window.mouse_passthrough_polygon = []
```

C#

```
// Set region, using Path2D node.
GetNode<Window>("Window").MousePassthrough = GetNode<Path2D>
("Path2D").Curve.GetBakedPoints();

// Set region, using Polygon2D node.
GetNode<Window>("Window").MousePassthrough = GetNode<Polygon2D>
("Polygon2D").Polygon;

// Reset region to default.
GetNode<Window>("Window").MousePassthrough = new Vector2[] {};
```

Note: This property is ignored if mouse_passthrough is set to `true`.

Note: On Windows, the portion of a window that lies outside the region is not drawn, while on Linux (X11) and macOS it is.

Note: This property is implemented on Linux (X11), macOS and Windows.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector2Array for more details.

---

bool **popup_window** = `false`

- void **set_flag**(flag: Flags, enabled: bool)
- bool **get_flag**(flag: Flags) const

If `true`, the **Window** will be considered a popup. Popups are sub-windows that don't show as separate windows in system's window manager's window list and will send close request when anything is clicked outside of them (unless exclusive is enabled).

Vector2i **position** = `Vector2i(0, 0)`

- void **set_position**(value: Vector2i)
- Vector2i **get_position**()

The window's position in pixels.

If ProjectSettings.display/window/subwindows/embed_subwindows is `false`, the position is in absolute screen coordinates. This typically applies to editor plugins. If the setting is `true`, the window's position is in the coordinates of its parent Viewport.

Note: This property only works if initial_position is set to WINDOW_INITIAL_POSITION_ABSOLUTE.

Vector2i **size** = `Vector2i(100, 100)` 🔗

- void **set_size**(value: Vector2i)
- Vector2i **get_size**()

The window's size in pixels.

Theme **theme** 🔗

- void **set_theme**(value: Theme)
- Theme **get_theme**()

The Theme resource this node and all its Control and **Window** children use. If a child node has its own Theme resource set, theme items are merged with child's definitions having higher priority.

Note: **Window** styles will have no effect unless the window is embedded.

StringName **theme_type_variation** = `&""` 🔗

- void **set_theme_type_variation**(value: StringName)
- StringName **get_theme_type_variation**()

The name of a theme type variation used by this **Window** to look up its own theme items. See Control.theme_type_variation for more details.

String **title** = `""` 🔗

- void **set_title**(value: String)
- String **get_title**()

The window's title. If the **Window** is native, title styles set in Theme will have no effect.

---

bool **transient** = `false` 🔗

- void **set_transient**(value: bool)
- bool **is_transient**()

If `true`, the **Window** is transient, i.e. it's considered a child of another **Window**. The transient window will be destroyed with its transient parent and will return focus to their parent when closed. The transient window is displayed on top of a non-exclusive full-screen parent window. Transient windows can't enter full-screen mode.

Note that behavior might be different depending on the platform.

---

bool **transient_to_focused** = `false` 🔗

- void **set_transient_to_focused**(value: bool)
- bool **is_transient_to_focused**()

If `true`, and the **Window** is transient, this window will (at the time of becoming visible) become transient to the currently focused window instead of the immediate parent window in the hierarchy. Note that the transient parent is assigned at the time this window becomes visible, so changing it afterwards has no effect until re-shown.

---

bool **transparent** = `false` 🔗

- void **set_flag**(flag: Flags, enabled: bool)
- bool **get_flag**(flag: Flags) const

If `true`, the **Window**'s background can be transparent. This is best used with embedded windows.

Note: Transparency support is implemented on Linux, macOS and Windows, but availability might vary depending on GPU driver, display manager, and compositor capabilities.

Note: This property has no effect if ProjectSettings.display/window/per_pixel_transparency/allowed is set to `false`.

---

bool **unfocusable** = `false` 🔗

- void **set_flag**(flag: Flags, enabled: bool)
- bool **get_flag**(flag: Flags) const

If `true`, the **Window** can't be focused nor interacted with. It can still be visible.

---

bool **unresizable** = `false` 🔗

- void **set_flag**(flag: Flags, enabled: bool)
- bool **get_flag**(flag: Flags) const

If `true`, the window can't be resized. Minimize and maximize buttons are disabled.

---

bool **visible** = `true` 🔗

- void **set_visible**(value: bool)
- bool **is_visible**()

If `true`, the window is visible.

---

bool **wrap_controls** = `false` 🔗

- void **set_wrap_controls**(value: bool)
- bool **is_wrapping_controls**()

If `true`, the window's size will automatically update when a child node is added or removed, ignoring min_size if the new size is bigger.

If `false`, you need to call child_controls_changed manually.

Method Descriptions

Vector2 **_get_contents_minimum_size**() virtual const

Virtual method to be implemented by the user. Overrides the value returned by get_contents_minimum_size.

---

void **add_theme_color_override**(name: StringName, color: Color)

Creates a local override for a theme Color with the specified `name`. Local overrides always take precedence when fetching theme items for the control. An override can be removed with remove_theme_color_override.

See also get_theme_color and Control.add_theme_color_override for more details.

---

void **add_theme_constant_override**(name: StringName, constant: int)

Creates a local override for a theme constant with the specified `name`. Local overrides always take precedence when fetching theme items for the control. An override can be removed with remove_theme_constant_override.

See also get_theme_constant.

---

void **add_theme_font_override**(name: StringName, font: Font)

Creates a local override for a theme Font with the specified `name`. Local overrides always take precedence when fetching theme items for the control. An override can be removed with remove_theme_font_override.

See also get_theme_font.

---

void **add_theme_font_size_override**(name: StringName, font_size: int) 🔗

Creates a local override for a theme font size with the specified `name`. Local overrides always take precedence when fetching theme items for the control. An override can be removed with remove_theme_font_size_override.

See also get_theme_font_size.

void **add_theme_icon_override**(name: StringName, texture: Texture2D) 🔗

Creates a local override for a theme icon with the specified `name`. Local overrides always take precedence when fetching theme items for the control. An override can be removed with remove_theme_icon_override.

See also get_theme_icon.

void **add_theme_stylebox_override**(name: StringName, stylebox: StyleBox) 🔗

Creates a local override for a theme StyleBox with the specified `name`. Local overrides always take precedence when fetching theme items for the control. An override can be removed with remove_theme_stylebox_override.

See also get_theme_stylebox and Control.add_theme_stylebox_override for more details.

void **begin_bulk_theme_override**() 🔗

Prevents `*_theme_*_override` methods from emitting NOTIFICATION_THEME_CHANGED until end_bulk_theme_override is called.

bool **can_draw**() const 🔗

Returns whether the window is being drawn to the screen.

---

void **child_controls_changed**() 🔗

Requests an update of the **Window** size to fit underlying Control nodes.

---

void **end_bulk_theme_override**() 🔗

Ends a bulk theme override update. See begin_bulk_theme_override.

Vector2 **get_contents_minimum_size**() const 🔗

Returns the combined minimum size from the child Control nodes of the window. Use child_controls_changed to update it when child nodes have changed.

The value returned by this method can be overridden with _get_contents_minimum_size.

---

bool **get_flag**(flag: Flags) const 🔗

Returns `true` if the `flag` is set.

LayoutDirection **get_layout_direction**() const 🔗

Returns layout direction and text writing direction.

Vector2i **get_position_with_decorations**() const 🔗

Returns the window's position including its border.

Note: If visible is `false`, this method returns the same value as position.

Vector2i **get_size_with_decorations**() const

Returns the window's size including its border.

Note: If visible is `false`, this method returns the same value as size.

Color **get_theme_color**(name: StringName, theme_type: StringName = &"") const

Returns a Color from the first matching Theme in the tree if that Theme has a color item with the specified `name` and `theme_type`.

See Control.get_theme_color for more details.

---

int **get_theme_constant**(name: StringName, theme_type: StringName = &"") const

Returns a constant from the first matching Theme in the tree if that Theme has a constant item with the specified `name` and `theme_type`.

See Control.get_theme_color for more details.

---

float **get_theme_default_base_scale**() const

Returns the default base scale value from the first matching Theme in the tree if that Theme has a valid Theme.default_base_scale value.

See Control.get_theme_color for details.

Font **get_theme_default_font**() const

Returns the default font from the first matching Theme in the tree if that Theme has a valid Theme.default_font value.

See Control.get_theme_color for details.

---

int **get_theme_default_font_size**() const 🔗

Returns the default font size value from the first matching Theme in the tree if that Theme has a valid Theme.default_font_size value.

See Control.get_theme_color for details.

Font **get_theme_font**(name: StringName, theme_type: StringName = &"") const 🔗

Returns a Font from the first matching Theme in the tree if that Theme has a font item with the specified `name` and `theme_type`.

See Control.get_theme_color for details.

---

int **get_theme_font_size**(name: StringName, theme_type: StringName = &"") const 🔗

Returns a font size from the first matching Theme in the tree if that Theme has a font size item with the specified `name` and `theme_type`.

See Control.get_theme_color for details.

Texture2D **get_theme_icon**(name: StringName, theme_type: StringName = &"") const 🔗

Returns an icon from the first matching Theme in the tree if that Theme has an icon item with the specified `name` and `theme_type`.

See Control.get_theme_color for details.

StyleBox **get_theme_stylebox**(name: StringName, theme_type: StringName = &"") const 🔗

Returns a StyleBox from the first matching Theme in the tree if that Theme has a stylebox item with the specified `name` and `theme_type`.

See Control.get_theme_color for details.

---

int **get_window_id**() const 🔗

Returns the ID of the window.

---

void **grab_focus**() 🔗

Causes the window to grab focus, allowing it to receive user input.

---

bool **has_focus**() const 🔗

Returns `true` if the window is focused.

---

bool **has_theme_color**(name: StringName, theme_type: StringName = &"") const 🔗

Returns `true` if there is a matching Theme in the tree that has a color item with the specified `name` and `theme_type`.

See Control.get_theme_color for details.

---

bool **has_theme_color_override**(name: StringName) const 🔗

Returns `true` if there is a local override for a theme Color with the specified `name` in this Control node.

See add_theme_color_override.

---

bool **has_theme_constant**(name: StringName, theme_type: StringName = &"") const 🔗

Returns `true` if there is a matching Theme in the tree that has a constant item with the specified `name` and `theme_type`.

See Control.get_theme_color for details.

---

bool **has_theme_constant_override**(name: StringName) const

Returns `true` if there is a local override for a theme constant with the specified `name` in this Control node.

See add_theme_constant_override.

---

bool **has_theme_font**(name: StringName, theme_type: StringName = &"") const

Returns `true` if there is a matching Theme in the tree that has a font item with the specified `name` and `theme_type`.

See Control.get_theme_color for details.

---

bool **has_theme_font_override**(name: StringName) const

Returns `true` if there is a local override for a theme Font with the specified `name` in this Control node.

See add_theme_font_override.

---

bool **has_theme_font_size**(name: StringName, theme_type: StringName = &"") const

Returns `true` if there is a matching Theme in the tree that has a font size item with the specified `name` and `theme_type`.

See Control.get_theme_color for details.

---

bool **has_theme_font_size_override**(name: StringName) const

Returns `true` if there is a local override for a theme font size with the specified `name` in this Control node.

See add_theme_font_size_override.

bool **has_theme_icon**(name: StringName, theme_type: StringName = &"") const

Returns `true` if there is a matching Theme in the tree that has an icon item with the specified `name` and `theme_type`.

See Control.get_theme_color for details.

bool **has_theme_icon_override**(name: StringName) const

Returns `true` if there is a local override for a theme icon with the specified `name` in this Control node.

See add_theme_icon_override.

bool **has_theme_stylebox**(name: StringName, theme_type: StringName = &"") const

Returns `true` if there is a matching Theme in the tree that has a stylebox item with the specified `name` and `theme_type`.

See Control.get_theme_color for details.

bool **has_theme_stylebox_override**(name: StringName) const

Returns `true` if there is a local override for a theme StyleBox with the specified `name` in this Control node.

See add_theme_stylebox_override.

void **hide**()

Hides the window. This is not the same as minimized state. Hidden window can't be interacted with and needs to be made visible with show.

---

bool **is_embedded**() const

Returns `true` if the window is currently embedded in another window.

---

bool **is_layout_rtl**() const

Returns `true` if layout is right-to-left.

---

bool **is_maximize_allowed**() const

Returns `true` if the window can be maximized (the maximize button is enabled).

---

bool **is_using_font_oversampling**() const

Returns `true` if font oversampling is enabled. See set_use_font_oversampling.

---

void **move_to_center**()

Centers a native window on the current screen and an embedded window on its embedder Viewport.

---

void **move_to_foreground**()

Deprecated: Use grab_focus instead.

Causes the window to grab focus, allowing it to receive user input.

---

void **popup**(rect: Rect2i = Rect2i(0, 0, 0, 0)) 🔗

Shows the **Window** and makes it transient (see transient). If `rect` is provided, it will be set as the **Window**'s size. Fails if called on the main window.

If ProjectSettings.display/window/subwindows/embed_subwindows is `true` (single-window mode), `rect`'s coordinates are global and relative to the main window's top-left corner (excluding window decorations). If `rect`'s position coordinates are negative, the window will be located outside the main window and may not be visible as a result.

If ProjectSettings.display/window/subwindows/embed_subwindows is `false` (multi-window mode), `rect`'s coordinates are global and relative to the top-left corner of the leftmost screen. If `rect`'s position coordinates are negative, the window will be placed at the top-left corner of the screen.

Note: `rect` must be in global coordinates if specified.

---

void **popup_centered**(minsize: Vector2i = Vector2i(0, 0)) 🔗

Popups the **Window** at the center of the current screen, with optionally given minimum size. If the **Window** is embedded, it will be centered in the parent Viewport instead.

Note: Calling it with the default value of `minsize` is equivalent to calling it with size.

---

void **popup_centered_clamped**(minsize: Vector2i = Vector2i(0, 0), fallback_ratio: float = 0.75) 🔗

Popups the **Window** centered inside its parent **Window**. `fallback_ratio` determines the maximum size of the **Window**, in relation to its parent.

Note: Calling it with the default value of `minsize` is equivalent to calling it with size.

---

void **popup_centered_ratio**(ratio: float = 0.8)

If **Window** is embedded, popups the **Window** centered inside its embedder and sets its size as a `ratio` of embedder's size.

If **Window** is a native window, popups the **Window** centered inside the screen of its parent **Window** and sets its size as a `ratio` of the screen size.

void **popup_exclusive**(from_node: Node, rect: Rect2i = Rect2i(0, 0, 0, 0))

Attempts to parent this dialog to the last exclusive window relative to `from_node`, and then calls popup on it. The dialog must have no current parent, otherwise the method fails.

See also set_unparent_when_invisible and Node.get_last_exclusive_window.

void **popup_exclusive_centered**(from_node: Node, minsize: Vector2i = Vector2i(0, 0))

Attempts to parent this dialog to the last exclusive window relative to `from_node`, and then calls popup_centered on it. The dialog must have no current parent, otherwise the method fails.

See also set_unparent_when_invisible and Node.get_last_exclusive_window.

void **popup_exclusive_centered_clamped**(from_node: Node, minsize: Vector2i = Vector2i(0, 0), fallback_ratio: float = 0.75)

Attempts to parent this dialog to the last exclusive window relative to `from_node`, and then calls popup_centered_clamped on it. The dialog must have no current parent, otherwise the method fails.

See also set_unparent_when_invisible and Node.get_last_exclusive_window.

void **popup_exclusive_centered_ratio**(from_node: Node, ratio: float = 0.8)

Attempts to parent this dialog to the last exclusive window relative to `from_node`, and then calls popup_centered_ratio on it. The dialog must have no current parent, otherwise the method fails.

See also set_unparent_when_invisible and Node.get_last_exclusive_window.

void **popup_exclusive_on_parent**(from_node: Node, parent_rect: Rect2i)

Attempts to parent this dialog to the last exclusive window relative to `from_node`, and then calls popup_on_parent on it. The dialog must have no current parent, otherwise the method fails.

See also set_unparent_when_invisible and Node.get_last_exclusive_window.

void **popup_on_parent**(parent_rect: Rect2i)

Popups the **Window** with a position shifted by parent **Window**'s position. If the **Window** is embedded, has the same effect as popup.

void **remove_theme_color_override**(name: StringName)

Removes a local override for a theme Color with the specified `name` previously added by add_theme_color_override or via the Inspector dock.

void **remove_theme_constant_override**(name: StringName)

Removes a local override for a theme constant with the specified `name` previously added by add_theme_constant_override or via the Inspector dock.

void **remove_theme_font_override**(name: StringName) 🔗

Removes a local override for a theme Font with the specified `name` previously added by add_theme_font_override or via the Inspector dock.

void **remove_theme_font_size_override**(name: StringName) 🔗

Removes a local override for a theme font size with the specified `name` previously added by add_theme_font_size_override or via the Inspector dock.

void **remove_theme_icon_override**(name: StringName) 🔗

Removes a local override for a theme icon with the specified `name` previously added by add_theme_icon_override or via the Inspector dock.

void **remove_theme_stylebox_override**(name: StringName) 🔗

Removes a local override for a theme StyleBox with the specified `name` previously added by add_theme_stylebox_override or via the Inspector dock.

void **request_attention**() 🔗

Tells the OS that the **Window** needs an attention. This makes the window stand out in some way depending on the system, e.g. it might blink on the task bar.

void **reset_size**() 🔗

Resets the size to the minimum size, which is the max of min_size and (if wrap_controls is enabled) get_contents_minimum_size. This is equivalent to calling `set_size(Vector2i())` (or any size below the minimum).

void **set_flag**(flag: Flags, enabled: bool)

Sets a specified window flag.

---

void **set_ime_active**(active: bool)

If `active` is `true`, enables system's native IME (Input Method Editor).

---

void **set_ime_position**(position: Vector2i)

Moves IME to the given position.

---

void **set_layout_direction**(direction: LayoutDirection)

Sets layout direction and text writing direction. Right-to-left layouts are necessary for certain languages (e.g. Arabic and Hebrew).

---

void **set_unparent_when_invisible**(unparent: bool)

If `unparent` is `true`, the window is automatically unparented when going invisible.

Note: Make sure to keep a reference to the node, otherwise it will be orphaned. You also need to manually call Node.queue_free to free the window if it's not parented.

---

void **set_use_font_oversampling**(enable: bool)

Enables font oversampling. This makes fonts look better when they are scaled up.

---

void **show**()

Makes the **Window** appear. This enables interactions with the **Window** and doesn't change any of its property other than visibility (unlike e.g. popup).

Theme Property Descriptions

Color **title_color** = `Color(0.875, 0.875, 0.875, 1)`

The color of the title's text.

Color **title_outline_modulate** = `Color(0, 0, 0, 1)`

The color of the title's text outline.

int **close_h_offset** = `18`

Horizontal position offset of the close button.

int **close_v_offset** = `24`

Vertical position offset of the close button.

int **resize_margin** = `4`

Defines the outside margin at which the window border can be grabbed with mouse and resized.

int **title_height** = `36`

Height of the title bar.

int **title_outline_size** = `0`

The size of the title outline.

Font **title_font** 🔗

The font used to draw the title.

int **title_font_size** 🔗

The size of the title font.

Texture2D **close** 🔗

The icon for the close button.

Texture2D **close_pressed** 🔗

The icon for the close button when it's being pressed.

StyleBox **embedded_border** 🔗

The background style used when the **Window** is embedded. Note that this is drawn only under the window's content, excluding the title. For proper borders and title bar style, you can use `expand_margin_*` properties of StyleBoxFlat.

Note: The content background will not be visible unless transparent is enabled.

StyleBox **embedded_unfocused_border** 🔗

The background style used when the **Window** is embedded and unfocused.

WorldEnvironment

Inherits: Node **<** Object

Default environment properties for the entire scene (post-processing effects, lighting and background settings).

Description

The **WorldEnvironment** node is used to configure the default Environment for the scene.

The parameters defined in the **WorldEnvironment** can be overridden by an Environment node set on the current Camera3D. Additionally, only one **WorldEnvironment** may be instantiated in a given scene at a time.

The **WorldEnvironment** allows the user to specify default lighting parameters (e.g. ambient lighting), various post-processing effects (e.g. SSAO, DOF, Tonemapping), and how to draw the background (e.g. solid color, skybox). Usually, these are added in order to improve the realism/color balance of the scene.

Tutorials

- Environment and post-processing
- 3D Material Testers Demo [https://godotengine.org/asset-library/asset/2742]
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

CameraAttributes	camera_attributes
Compositor	compositor
Environment	environment

Property Descriptions

CameraAttributes **camera_attributes** 🔗

- void **set_camera_attributes**(value: CameraAttributes)
- CameraAttributes **get_camera_attributes**()

The default CameraAttributes resource to use if none set on the Camera3D.

Compositor **compositor** 🔗

- void **set_compositor**(value: Compositor)
- Compositor **get_compositor**()

The default Compositor resource to use if none set on the Camera3D.

Environment **environment** 🔗

- void **set_environment**(value: Environment)

- Environment **get_environment**()

The Environment resource used by this **WorldEnvironment**, defining the default properties.

XRAnchor3D

Inherits: XRNode3D **<** Node3D **<** Node **<** Object

An anchor point in AR space.

Description

The **XRAnchor3D** point is an XRNode3D that maps a real world location identified by the AR platform to a position within the game world. For example, as long as plane detection in ARKit is on, ARKit will identify and update the position of planes (tables, floors, etc.) and create anchors for them.

This node is mapped to one of the anchors through its unique ID. When you receive a signal that a new anchor is available, you should add this node to your scene for that anchor. You can predefine nodes and set the ID; the nodes will simply remain on 0,0,0 until a plane is recognized.

Keep in mind that, as long as plane detection is enabled, the size, placing and orientation of an anchor will be updated as the detection logic learns more about the real world out there especially if only part of the surface is in view.

Tutorials

- XR documentation index

Methods

Plane	get_plane() const
Vector3	get_size() const

Method Descriptions

Plane **get_plane**() const

Returns a plane aligned with our anchor; handy for intersection testing.

Vector3 **get_size**() const

Returns the estimated size of the plane that was detected. Say when the anchor relates to a table in the real world, this is the estimated size of the surface of that table.

XRBodyModifier3D

Experimental: This class may be changed or removed in future versions.

Inherits: SkeletonModifier3D **<** Node3D **<** Node **<** Object

A node for driving body meshes from XRBodyTracker data.

Description

This node uses body tracking data from an XRBodyTracker to pose the skeleton of a body mesh.

Positioning of the body is performed by creating an XRNode3D ancestor of the body mesh driven by the same XRBodyTracker.

The body tracking position-data is scaled by Skeleton3D.motion_scale when applied to the skeleton, which can be used to adjust the tracked body to match the scale of the body model.

Tutorials

- XR documentation index

Properties

StringName	body_tracker	`&"/user/body_tracker"`

BitField[BodyUpdate]	body_update	`7`
BoneUpdate	bone_update	`0`

Enumerations

flags **BodyUpdate**: 🔗

BodyUpdate **BODY_UPDATE_UPPER_BODY** = `1`

The skeleton's upper body joints are updated.

BodyUpdate **BODY_UPDATE_LOWER_BODY** = `2`

The skeleton's lower body joints are updated.

BodyUpdate **BODY_UPDATE_HANDS** = `4`

The skeleton's hand joints are updated.

enum **BoneUpdate**: 🔗

BoneUpdate **BONE_UPDATE_FULL** = `0`

The skeleton's bones are fully updated (both position and rotation) to match the tracked bones.

BoneUpdate **BONE_UPDATE_ROTATION_ONLY** = `1`

The skeleton's bones are only rotated to align with the tracked bones, preserving bone length.

BoneUpdate **BONE_UPDATE_MAX** = `2`

Represents the size of the BoneUpdate enum.

Property Descriptions

StringName **body_tracker** = `&"/user/body_tracker"`

- void **set_body_tracker**(value: StringName)
- StringName **get_body_tracker**()

The name of the XRBodyTracker registered with XRServer to obtain the body tracking data from.

BitField[BodyUpdate] **body_update** = `7`

- void **set_body_update**(value: BitField[BodyUpdate])
- BitField[BodyUpdate] **get_body_update**()

Specifies the body parts to update.

BoneUpdate **bone_update** = `0`

- void **set_bone_update**(value: BoneUpdate)
- BoneUpdate **get_bone_update**()

Specifies the type of updates to perform on the bones.

XRCamera3D

Inherits: Camera3D **<** Node3D **<** Node **<** Object

A camera node with a few overrules for AR/VR applied, such as location tracking.

Description

This is a helper spatial node for our camera; note that, if stereoscopic rendering is applicable (VR-HMD), most of the camera properties are ignored, as the HMD information overrides them. The only properties that can be trusted are the near and far planes.

The position and orientation of this node is automatically updated by the XR Server to represent the location of the HMD if such tracking is available and can thus be used by game logic. Note that, in contrast to the XR Controller, the render thread has access to the most up-to-date tracking data of the HMD and the location of the XRCamera3D can lag a few milliseconds behind what is used for rendering as a result.

Tutorials

- XR documentation index

XRController3D

Inherits: XRNode3D **<** Node3D **<** Node **<** Object

A spatial node representing a spatially-tracked controller.

Description

This is a helper spatial node that is linked to the tracking of controllers. It also offers several handy passthroughs to the state of buttons and such on the controllers.

Controllers are linked by their ID. You can create controller nodes before the controllers are available. If your game always uses two controllers (one for each hand), you can predefine the controllers with ID 1 and 2; they will become active as soon as the controllers are identified. If you expect additional controllers to be used, you should react to the signals and add XRController3D nodes to your scene.

The position of the controller node is automatically updated by the XRServer. This makes this node ideal to add child nodes to visualize the controller.

As many XR runtimes now use a configurable action map all inputs are named.

Tutorials

- XR documentation index

Methods

float	get_float(name: StringName) const
Variant	get_input(name: StringName) const
TrackerHand	get_tracker_hand() const
Vector2	get_vector2(name: StringName) const
bool	is_button_pressed(name: StringName) const

Signals

button_pressed(name: String) 🔗

Emitted when a button on this controller is pressed.

button_released(name: String) 🔗

Emitted when a button on this controller is released.

input_float_changed(name: String, value: float) 🔗

Emitted when a trigger or similar input on this controller changes value.

input_vector2_changed(name: String, value: Vector2) 🔗

Emitted when a thumbstick or thumbpad on this controller is moved.

---

profile_changed(role: String) 🔗

Emitted when the interaction profile on this controller is changed.

Method Descriptions

float **get_float**(name: StringName) const 🔗

Returns a numeric value for the input with the given `name`. This is used for triggers and grip sensors.

Variant **get_input**(name: StringName) const 🔗

Returns a Variant for the input with the given `name`. This works for any input type, the variant will be typed according to the actions configuration.

TrackerHand **get_tracker_hand**() const 🔗

Returns the hand holding this controller, if known. See TrackerHand.

Vector2 **get_vector2**(name: StringName) const 🔗

Returns a Vector2 for the input with the given `name`. This is used for thumbsticks and thumbpads found on many controllers.

---

bool **is_button_pressed**(name: StringName) const 🔗

Returns `true` if the button with the given `name` is pressed.

XRFaceModifier3D

Experimental: This class may be changed or removed in future versions.

Inherits: Node3D **<** Node **<** Object

A node for driving standard face meshes from XRFaceTracker weights.

Description

This node applies weights from a XRFaceTracker to a mesh with supporting face blend shapes.

The Unified Expressions [https://docs.vrcft.io/docs/tutorial-avatars/tutorial-avatars-extras/unified-blendshapes] blend shapes are supported, as well as ARKit and SRanipal blend shapes.

The node attempts to identify blend shapes based on name matching. Blend shapes should match the names listed in the Unified Expressions Compatibility [https://docs.vrcft.io/docs/tutorial-avatars/tutorial-avatars-extras/compatibility/overview] chart.

Tutorials

- XR documentation index

Properties

StringName	face_tracker	`&"/user/face_tracker"`
NodePath	target	`NodePath("")`

Property Descriptions

StringName **face_tracker** = `&"/user/face_tracker"`

- void **set_face_tracker**(value: StringName)
- StringName **get_face_tracker**()

The XRFaceTracker path.

NodePath **target** = `NodePath("")`

- void **set_target**(value: NodePath)
- NodePath **get_target**()

The NodePath of the face MeshInstance3D.

XRHandModifier3D

Inherits: SkeletonModifier3D **<** Node3D **<** Node **<** Object

A node for driving hand meshes from XRHandTracker data.

Description

This node uses hand tracking data from an XRHandTracker to pose the skeleton of a hand mesh.

Positioning of hands is performed by creating an XRNode3D ancestor of the hand mesh driven by the same XRHandTracker.

The hand tracking position-data is scaled by Skeleton3D.motion_scale when applied to the skeleton, which can be used to adjust the tracked hand to match the scale of the hand model.

Tutorials

- XR documentation index

Properties

BoneUpdate	bone_update	`0`
StringName	hand_tracker	`&"/user/hand_tracker/left"`

Enumerations

enum **BoneUpdate**:

BoneUpdate **BONE_UPDATE_FULL** = `0`

The skeleton's bones are fully updated (both position and rotation) to match the tracked bones.

BoneUpdate **BONE_UPDATE_ROTATION_ONLY** = `1`

The skeleton's bones are only rotated to align with the tracked bones, preserving bone length.

BoneUpdate **BONE_UPDATE_MAX** = `2`

Represents the size of the BoneUpdate enum.

Property Descriptions

BoneUpdate **bone_update** = `0`

- void **set_bone_update**(value: BoneUpdate)
- BoneUpdate **get_bone_update**()

Specifies the type of updates to perform on the bones.

StringName **hand_tracker** = `&"/user/hand_tracker/left"`

- void **set_hand_tracker**(value: StringName)
- StringName **get_hand_tracker**()

The name of the XRHandTracker registered with XRServer to obtain the hand tracking data from.

XRNode3D

Inherits: Node3D **<** Node **<** Object

Inherited By: XRAnchor3D, XRController3D

A spatial node that has its position automatically updated by the XRServer.

Description

This node can be bound to a specific pose of a XRPositionalTracker and will automatically have its Node3D.transform updated by the XRServer. Nodes of this type must be added as children of the XROrigin3D node.

Tutorials

- XR documentation index

Properties

StringName	pose	`&"default"`
bool	show_when_tracked	`false`
StringName	tracker	`&""`

Methods

bool	get_has_tracking_data() const
bool	get_is_active() const
XRPose	get_pose()
void	trigger_haptic_pulse(action_name: String, frequency: float, amplitude: float, duration_sec: float, delay_sec: float)

Signals

tracking_changed(tracking: bool)

Emitted when the tracker starts or stops receiving updated tracking data for the pose being tracked. The `tracking` argument indicates whether the tracker is getting updated tracking data.

Property Descriptions

StringName **pose** = `&"default"`

- void **set_pose_name**(value: StringName)
- StringName **get_pose_name**()

The name of the pose we're bound to. Which poses a tracker supports is not known during design time.

Godot defines number of standard pose names such as `aim` and `grip` but other may be configured within a given XRInterface.

---

bool **show_when_tracked** = `false`

- void **set_show_when_tracked**(value: bool)
- bool **get_show_when_tracked**()

Enables showing the node when tracking starts, and hiding the node when tracking is lost.

StringName **tracker** = `&""`

- void **set_tracker**(value: StringName)
- StringName **get_tracker**()

The name of the tracker we're bound to. Which trackers are available is not known during design time.

Godot defines a number of standard trackers such as `left_hand` and `right_hand` but others may be configured within a given XRInterface.

Method Descriptions

bool **get_has_tracking_data**() const

Returns `true` if the tracker has current tracking data for the pose being tracked.

---

bool **get_is_active**() const

Returns `true` if the tracker has been registered and the pose is being tracked.

XRPose **get_pose**()

Returns the XRPose containing the current state of the pose being tracked. This gives access to additional properties of this pose.

---

void **trigger_haptic_pulse**(action_name: String, frequency: float, amplitude: float, duration_sec: float, delay_sec: float)

Triggers a haptic pulse on a device associated with this interface.

`action_name` is the name of the action for this pulse.

`frequency` is the frequency of the pulse, set to `0.0` to have the system use a default frequency.

`amplitude` is the amplitude of the pulse between `0.0` and `1.0`.

`duration_sec` is the duration of the pulse in seconds.

`delay_sec` is a delay in seconds before the pulse is given.

XROrigin3D

Inherits: Node3D **<** Node **<** Object

The origin point in AR/VR.

Description

This is a special node within the AR/VR system that maps the physical location of the center of our tracking space to the virtual location within our game world.

Multiple origin points can be added to the scene tree, but only one can used at a time. All the XRCamera3D, XRController3D, and XRAnchor3D nodes should be direct children of this node for spatial tracking to work correctly.

It is the position of this node that you update when your character needs to move through your game world while we're not moving in the real world. Movement in the real world is always in relation to this origin point.

For example, if your character is driving a car, the **XROrigin3D** node should be a child node of this car. Or, if you're implementing a teleport system to move your character, you should change the position of this node.

Tutorials

- XR documentation index

Properties

bool	current	`false`
float	world_scale	`1.0`

Property Descriptions

bool **current** = `false` 🔗

- void **set_current**(value: bool)
- bool **is_current**()

If `true`, this origin node is currently being used by the XRServer. Only one origin point can be used at a time.

float **world_scale** = `1.0` 🔗

- void **set_world_scale**(value: float)
- float **get_world_scale**()

The scale of the game world compared to the real world. This is the same as XRServer.world_scale. By default, most AR/VR platforms assume that 1 game unit corresponds to 1 real world meter.

Resource

Inherits: RefCounted **<** Object

Inherited By: Animation, AnimationLibrary, AnimationNode, AnimationNodeStateMachinePlayback, AnimationNodeStateMachineTransition, AudioBusLayout, AudioEffect, AudioStream, BitMap, BoneMap, ButtonGroup, CameraAttributes, Compositor, CompositorEffect, CryptoKey, Curve, Curve2D, Curve3D, EditorNode3DGizmoPlugin, EditorSettings, Environment, Font, GDExtension, GLTFAccessor, GLTFAnimation, GLTFBufferView, GLTFCamera, GLTFDocument, GLTFDocumentExtension, GLTFLight, GLTFMesh, GLTFNode, GLTFPhysicsBody, GLTFPhysicsShape, GLTFSkeleton, GLTFSkin, GLTFSpecGloss, GLTFState, GLTFTexture, GLTFTextureSampler, Gradient, Image, ImporterMesh, InputEvent, JSON, LabelSettings, LightmapGIData, Material, Mesh, MeshLibrary, MissingResource, MultiMesh, NavigationMesh, NavigationMeshSourceGeometryData2D, NavigationMeshSourceGeometryData3D, NavigationPolygon, Noise, Occluder3D, OccluderPolygon2D, OggPacketSequence, OpenXRAction, OpenXRActionMap, OpenXRActionSet, OpenXRInteractionProfile, OpenXRIPBinding, PackedDataContainer, PackedScene, PhysicsMaterial, PolygonPathFinder, RDShaderFile, RDShaderSPIRV, RichTextEffect, SceneReplicationConfig, Script, Shader, ShaderInclude, Shape2D, Shape3D, Shortcut, SkeletonModification2D, SkeletonModificationStack2D, SkeletonProfile, Skin, Sky, SpriteFrames, StyleBox, SyntaxHighlighter, Texture, Theme, TileMapPattern, TileSet, TileSetSource, Translation, VideoStream, VideoStreamPlayback, VisualShaderNode, VoxelGIData, World2D, World3D, X509Certificate

Base class for serializable objects.

Description

Resource is the base class for all Godot-specific resource types, serving primarily as data containers. Since they inherit from RefCounted, resources are reference-counted and freed when no longer in use. They can also be nested within other resources, and saved on disk. PackedScene, one of the most common Objects in a Godot project, is also a resource, uniquely capable of storing and instantiating the Nodes it contains as many times as desired.

In GDScript, resources can loaded from disk by their resource_path using @GDScript.load or @GDScript.preload.

The engine keeps a global cache of all loaded resources, referenced by paths (see ResourceLoader.has_cached). A resource will be cached when loaded for the first time and removed from cache once all references are released. When a resource is cached, subsequent loads using its path will return the cached reference.

Note: In C#, resources will not be freed instantly after they are no longer in use. Instead, garbage collection will run periodically and will free resources that are no longer in use. This means that unused resources will remain in memory for a while before being removed.

Tutorials

- Resources
- When and how to avoid using nodes for everything

Properties

bool	resource_local_to_scene	`false`

String	resource_name	""
String	resource_path	""
String	resource_scene_unique_id	

Methods

RID	_get_rid() virtual
void	_setup_local_to_scene() virtual
Resource	duplicate(subresources: bool = false) const
void	emit_changed()
String	generate_scene_unique_id() static
Node	get_local_scene() const
RID	get_rid() const
void	setup_local_to_scene()

void	take_over_path(path: String)

Signals

changed() 🔗

Emitted when the resource changes, usually when one of its properties is modified. See also emit_changed.

Note: This signal is not emitted automatically for properties of custom resources. If necessary, a setter needs to be created to emit the signal.

setup_local_to_scene_requested() 🔗

Deprecated: This signal is only emitted when the resource is created. Override _setup_local_to_scene instead.

Emitted by a newly duplicated resource with resource_local_to_scene set to `true`.

Property Descriptions

bool **resource_local_to_scene** = `false` 🔗

- void **set_local_to_scene**(value: bool)
- bool **is_local_to_scene**()

If `true`, the resource is duplicated for each instance of all scenes using it. At run-time, the resource can be modified in one scene without affecting other instances (see PackedScene.instantiate).

Note: Changing this property at run-time has no effect on already created duplicate resources.

String **resource_name** = `""` 🔗

- void **set_name**(value: String)
- String **get_name**()

An optional name for this resource. When defined, its value is displayed to represent the resource in the Inspector dock. For built-in scripts, the name is displayed as part of the tab name in the script editor.

Note: Some resource formats do not support resource names. You can still set the name in the editor or via code, but it will be lost when the resource is reloaded. For example, only built-in scripts can have a resource name, while scripts stored in separate files cannot.

String **resource_path** = `""` 🔗

- void **set_path**(value: String)
- String **get_path**()

The unique path to this resource. If it has been saved to disk, the value will be its filepath. If the resource is exclusively contained within a scene, the value will be the PackedScene's filepath, followed by a unique identifier.

Note: Setting this property manually may fail if a resource with the same path has already been previously loaded. If necessary, use take_over_path.

String **resource_scene_unique_id**

- void **set_scene_unique_id**(value: String)
- String **get_scene_unique_id**()

An unique identifier relative to the this resource's scene. If left empty, the ID is automatically generated when this resource is saved inside a PackedScene. If the resource is not inside a scene, this property is empty by default.

Note: When the PackedScene is saved, if multiple resources in the same scene use the same ID, only the earliest resource in the scene hierarchy keeps the original ID. The other resources are assigned new IDs from generate_scene_unique_id.

Note: Setting this property does not emit the changed signal.

Warning: When setting, the ID must only consist of letters, numbers, and underscores. Otherwise, it will fail and default to a randomly generated ID.

Method Descriptions

RID **_get_rid**() virtual

Override this method to return a custom RID when get_rid is called.

---

void **_setup_local_to_scene**() virtual

Override this method to customize the newly duplicated resource created from PackedScene.instantiate, if the original's resource_local_to_scene is set to `true`.

Example: Set a random `damage` value to every local resource from an instantiated scene.

```
extends Resource

var damage = 0

func _setup_local_to_scene():
    damage = randi_range(10, 40)
```

Resource **duplicate**(subresources: bool = false) const 🔗

Duplicates this resource, returning a new resource with its `export`ed or @GlobalScope.PROPERTY_USAGE_STORAGE properties copied from the original.

If `subresources` is `false`, a shallow copy is returned; nested resources within subresources are not duplicated and are shared with the original resource (with one exception; see below). If `subresources` is `true`, a deep copy is returned; nested subresources will be duplicated and are not shared (with two exceptions; see below).

`subresources` is usually respected, with the following exceptions:

- Subresource properties with the @GlobalScope.PROPERTY_USAGE_ALWAYS_DUPLICATE flag are always duplicated.
- Subresource properties with the @GlobalScope.PROPERTY_USAGE_NEVER_DUPLICATE flag are never duplicated.
- Subresources inside Array and Dictionary properties are never duplicated.

Note: For custom resources, this method will fail if Object._init has been defined with required parameters.

---

void **emit_changed**() 🔗

Emits the changed signal. This method is called automatically for some built-in resources.

Note: For custom resources, it's recommended to call this method whenever a meaningful change occurs, such as a modified property. This ensures that custom Objects depending on the resource are properly updated.

```
var damage:
    set(new_value):
        if damage != new_value:
            damage = new_value
            emit_changed()
```

String **generate_scene_unique_id**() static 🔗

Generates a unique identifier for a resource to be contained inside a PackedScene, based on the current date, time, and a random value. The returned string is only composed of letters (`a` to `y`) and numbers (`0` to `8`). See also resource_scene_unique_id.

Node **get_local_scene**() const 🔗

If resource_local_to_scene is set to `true` and the resource has been loaded from a PackedScene instantiation, returns the root Node of the scene where this resource is used. Otherwise, returns `null`.

RID **get_rid**() const 🔗

Returns the RID of this resource (or an empty RID). Many resources (such as Texture2D, Mesh, and so on) are high-level abstractions of resources stored in a specialized server (DisplayServer, RenderingServer, etc.), so this function will return the original RID.

---

void **setup_local_to_scene**() 🔗

Deprecated: This method should only be called internally.

Calls _setup_local_to_scene. If resource_local_to_scene is set to `true`, this method is automatically called from PackedScene.instantiate by the newly duplicated resource within the scene instance.

---

void **take_over_path**(path: String) 🔗

Sets the resource_path to `path`, potentially overriding an existing cache entry for this path. Further attempts to load an overridden resource by path will instead return this resource.

AnimatedTexture

Deprecated: This class does not work properly in current versions and may be removed in the future. There is currently no equivalent workaround.

Inherits: Texture2D **<** Texture **<** Resource **<** RefCounted **<** Object

Proxy texture for simple frame-based animations.

Description

AnimatedTexture is a resource format for frame-based animations, where multiple textures can be chained automatically with a predefined delay for each frame. Unlike AnimationPlayer or AnimatedSprite2D, it isn't a Node, but has the advantage of being usable anywhere a Texture2D resource can be used, e.g. in a TileSet.

The playback of the animation is controlled by the speed_scale property, as well as each frame's duration (see set_frame_duration). The animation loops, i.e. it will restart at frame 0 automatically after playing the last frame.

AnimatedTexture currently requires all frame textures to have the same size, otherwise the bigger ones will be cropped to match the smallest one.

Note: AnimatedTexture doesn't support using AtlasTextures. Each frame needs to be a separate Texture2D.

Warning: The current implementation is not efficient for the modern renderers.

Properties

int	current_frame	
int	frames	`1`
bool	one_shot	`false`
bool	pause	`false`
bool	resource_local_to_scene	`false` (overrides Resource)
float	speed_scale	`1.0`

Methods

float	get_frame_duration(frame: int) const
Texture2D	get_frame_texture(frame: int) const
void	set_frame_duration(frame: int, duration: float)
void	set_frame_texture(frame: int, texture: Texture2D)

Constants

MAX_FRAMES = `256`

The maximum number of frames supported by **AnimatedTexture**. If you need more frames in your animation, use AnimationPlayer or AnimatedSprite2D.

Property Descriptions

int **current_frame**

- void **set_current_frame**(value: int)
- int **get_current_frame**()

Sets the currently visible frame of the texture. Setting this frame while playing resets the current frame time, so the newly selected frame plays for its whole configured frame duration.

---

int **frames** = `1`

- void **set_frames**(value: int)
- int **get_frames**()

Number of frames to use in the animation. While you can create the frames independently with set_frame_texture, you need to set this value for the animation to take new frames into account. The maximum number of frames is MAX_FRAMES.

---

bool **one_shot** = `false`

- void **set_one_shot**(value: bool)

- bool **get_one_shot**()

If `true`, the animation will only play once and will not loop back to the first frame after reaching the end. Note that reaching the end will not set pause to `true`.

---

bool **pause** = `false`

- void **set_pause**(value: bool)
- bool **get_pause**()

If `true`, the animation will pause where it currently is (i.e. at current_frame). The animation will continue from where it was paused when changing this property to `false`.

---

float **speed_scale** = `1.0`

- void **set_speed_scale**(value: float)
- float **get_speed_scale**()

The animation speed is multiplied by this value. If set to a negative value, the animation is played in reverse.

Method Descriptions

float **get_frame_duration**(frame: int) const

Returns the given `frame`'s duration, in seconds.

Texture2D **get_frame_texture**(frame: int) const

Returns the given frame's Texture2D.

---

void **set_frame_duration**(frame: int, duration: float) 🔗

Sets the duration of any given `frame`. The final duration is affected by the speed_scale. If set to `0`, the frame is skipped during playback.

---

void **set_frame_texture**(frame: int, texture: Texture2D) 🔗

Assigns a Texture2D to the given frame. Frame IDs start at 0, so the first frame has ID 0, and the last frame of the animation has ID frames - 1.

You can define any number of textures up to MAX_FRAMES, but keep in mind that only frames from 0 to frames - 1 will be part of the animation.

Animation

Inherits: Resource **<** RefCounted **<** Object

Holds data that can be used to animate anything in the engine.

Description

This resource holds data that can be used to animate anything in the engine. Animations are divided into tracks and each track must be linked to a node. The state of that node can be changed through time, by adding timed keys (events) to the track.

GDScript

```
# This creates an animation that makes the node "Enemy" move to the right by
# 100 pixels in 2.0 seconds.
var animation = Animation.new()
var track_index = animation.add_track(Animation.TYPE_VALUE)
animation.track_set_path(track_index, "Enemy:position:x")
animation.track_insert_key(track_index, 0.0, 0)
animation.track_insert_key(track_index, 2.0, 100)
animation.length = 2.0
```

C#

```
// This creates an animation that makes the node "Enemy" move to the right by
// 100 pixels in 2.0 seconds.
var animation = new Animation();
int trackIndex = animation.AddTrack(Animation.TrackType.Value);
animation.TrackSetPath(trackIndex, "Enemy:position:x");
animation.TrackInsertKey(trackIndex, 0.0f, 0);
animation.TrackInsertKey(trackIndex, 2.0f, 100);
animation.Length = 2.0f;
```

Animations are just data containers, and must be added to nodes such as an AnimationPlayer to be played back. Animation tracks have different types, each with its own set of dedicated methods. Check TrackType to see available types.

Note: For 3D position/rotation/scale, using the dedicated TYPE_POSITION_3D, TYPE_ROTATION_3D and TYPE_SCALE_3D track types instead of TYPE_VALUE is recommended for performance reasons.

Tutorials

- Animation documentation index

Properties

bool	capture_included	`false`
float	length	`1.0`
LoopMode	loop_mode	`0`
float	step	`0.0333333`

Methods

int	add_track(type: TrackType, at_position: int = -1)
StringName	animation_track_get_key_animation(track_idx: int, key_idx: int) const
int	animation_track_insert_key(track_idx: int, time: float, animation: StringName)
void	animation_track_set_key_animation(track_idx: int, key_idx: int, animation: StringName)
float	audio_track_get_key_end_offset(track_idx: int, key_idx: int) const
float	audio_track_get_key_start_offset(track_idx: int, key_idx: int) const
Resource	audio_track_get_key_stream(track_idx: int, key_idx: int) const
int	audio_track_insert_key(track_idx: int, time: float, stream: Resource, start_offset: float = 0, end_offset: float = 0)
bool	audio_track_is_use_blend(track_idx: int) const

void	audio_track_set_key_end_offset(track_idx: int, key_idx: int, offset: float)
void	audio_track_set_key_start_offset(track_idx: int, key_idx: int, offset: float)
void	audio_track_set_key_stream(track_idx: int, key_idx: int, stream: Resource)
void	audio_track_set_use_blend(track_idx: int, enable: bool)
Vector2	bezier_track_get_key_in_handle(track_idx: int, key_idx: int) const
Vector2	bezier_track_get_key_out_handle(track_idx: int, key_idx: int) const
float	bezier_track_get_key_value(track_idx: int, key_idx: int) const
int	bezier_track_insert_key(track_idx: int, time: float, value: float, in_handle: Vector2 = Vector2(0, 0), out_handle: Vector2 = Vector2(0, 0))
float	bezier_track_interpolate(track_idx: int, time: float) const

void	bezier_track_set_key_in_handle(track_idx: int, key_idx: int, in_handle: Vector2, balanced_value_time_ratio: float = 1.0)
void	bezier_track_set_key_out_handle(track_idx: int, key_idx: int, out_handle: Vector2, balanced_value_time_ratio: float = 1.0)
void	bezier_track_set_key_value(track_idx: int, key_idx: int, value: float)
int	blend_shape_track_insert_key(track_idx: int, time: float, amount: float)
float	blend_shape_track_interpolate(track_idx: int, time_sec: float, backward: bool = false) const
void	clear()
void	compress(page_size: int = 8192, fps: int = 120, split_tolerance: float = 4.0)
void	copy_track(track_idx: int, to_animation: Animation)
int	find_track(path: NodePath, type: TrackType) const

int	get_track_count() const
StringName	method_track_get_name(track_idx: int, key_idx: int) const
Array	method_track_get_params(track_idx: int, key_idx: int) const
int	position_track_insert_key(track_idx: int, time: float, position: Vector3)
Vector3	position_track_interpolate(track_idx: int, time_sec: float, backward: bool = false) const
void	remove_track(track_idx: int)
int	rotation_track_insert_key(track_idx: int, time: float, rotation: Quaternion)
Quaternion	rotation_track_interpolate(track_idx: int, time_sec: float, backward: bool = false) const
int	scale_track_insert_key(track_idx: int, time: float, scale: Vector3)
Vector3	scale_track_interpolate(track_idx: int, time_sec: float, backward: bool = false) const

int	track_find_key(track_idx: int, time: float, find_mode: FindMode = 0, limit: bool = false, backward: bool = false) const
bool	track_get_interpolation_loop_wrap(track_idx: int) const
InterpolationType	track_get_interpolation_type(track_idx: int) const
int	track_get_key_count(track_idx: int) const
float	track_get_key_time(track_idx: int, key_idx: int) const
float	track_get_key_transition(track_idx: int, key_idx: int) const
Variant	track_get_key_value(track_idx: int, key_idx: int) const
NodePath	track_get_path(track_idx: int) const
TrackType	track_get_type(track_idx: int) const
int	track_insert_key(track_idx: int, time: float, key: Variant, transition: float = 1)

bool	track_is_compressed(track_idx: int) const
bool	track_is_enabled(track_idx: int) const
bool	track_is_imported(track_idx: int) const
void	track_move_down(track_idx: int)
void	track_move_to(track_idx: int, to_idx: int)
void	track_move_up(track_idx: int)
void	track_remove_key(track_idx: int, key_idx: int)
void	track_remove_key_at_time(track_idx: int, time: float)
void	track_set_enabled(track_idx: int, enabled: bool)
void	track_set_imported(track_idx: int, imported: bool)
void	track_set_interpolation_loop_wrap(track_idx: int, interpolation: bool)

void	track_set_interpolation_type(track_idx: int, interpolation: InterpolationType)
void	track_set_key_time(track_idx: int, key_idx: int, time: float)
void	track_set_key_transition(track_idx: int, key_idx: int, transition: float)
void	track_set_key_value(track_idx: int, key: int, value: Variant)
void	track_set_path(track_idx: int, path: NodePath)
void	track_swap(track_idx: int, with_idx: int)
UpdateMode	value_track_get_update_mode(track_idx: int) const
Variant	value_track_interpolate(track_idx: int, time_sec: float, backward: bool = false) const
void	value_track_set_update_mode(track_idx: int, mode: UpdateMode)

Enumerations

enum **TrackType**: 🔗

TrackType **TYPE_VALUE** = `0`

Value tracks set values in node properties, but only those which can be interpolated. For 3D position/rotation/scale, using the dedicated TYPE_POSITION_3D, TYPE_ROTATION_3D and TYPE_SCALE_3D track types instead of TYPE_VALUE is recommended for performance reasons.

TrackType **TYPE_POSITION_3D** = `1`

3D position track (values are stored in Vector3s).

TrackType **TYPE_ROTATION_3D** = `2`

3D rotation track (values are stored in Quaternions).

TrackType **TYPE_SCALE_3D** = `3`

3D scale track (values are stored in Vector3s).

TrackType **TYPE_BLEND_SHAPE** = `4`

Blend shape track.

TrackType **TYPE_METHOD** = `5`

Method tracks call functions with given arguments per key.

TrackType **TYPE_BEZIER** = `6`

Bezier tracks are used to interpolate a value using custom curves. They can also be used to animate sub-properties of vectors and colors (e.g. alpha value of a Color).

TrackType **TYPE_AUDIO** = `7`

Audio tracks are used to play an audio stream with either type of AudioStreamPlayer. The stream can be trimmed and previewed in the animation.

TrackType **TYPE_ANIMATION** = `8`

Animation tracks play animations in other AnimationPlayer nodes.

---

enum **InterpolationType**: 🔗

InterpolationType **INTERPOLATION_NEAREST** = `0`

No interpolation (nearest value).

InterpolationType **INTERPOLATION_LINEAR** = `1`

Linear interpolation.

InterpolationType **INTERPOLATION_CUBIC** = `2`

Cubic interpolation. This looks smoother than linear interpolation, but is more expensive to interpolate. Stick to INTERPOLATION_LINEAR for complex 3D animations imported from external software, even if it requires using a higher animation framerate in return.

InterpolationType **INTERPOLATION_LINEAR_ANGLE** = `3`

Linear interpolation with shortest path rotation.

Note: The result value is always normalized and may not match the key value.

InterpolationType **INTERPOLATION_CUBIC_ANGLE** = `4`

Cubic interpolation with shortest path rotation.

Note: The result value is always normalized and may not match the key value.

---

enum **UpdateMode**: 🔗

UpdateMode **UPDATE_CONTINUOUS** = `0`

Update between keyframes and hold the value.

UpdateMode **UPDATE_DISCRETE** = `1`

Update at the keyframes.

UpdateMode **UPDATE_CAPTURE** = `2`

Same as UPDATE_CONTINUOUS but works as a flag to capture the value of the current object and perform interpolation in some methods. See also AnimationMixer.capture, AnimationPlayer.playback_auto_capture, and AnimationPlayer.play_with_capture.

---

enum **LoopMode**: 🔗

LoopMode **LOOP_NONE** = `0`

At both ends of the animation, the animation will stop playing.

LoopMode **LOOP_LINEAR** = `1`

At both ends of the animation, the animation will be repeated without changing the playback direction.

LoopMode **LOOP_PINGPONG** = `2`

Repeats playback and reverse playback at both ends of the animation.

---

enum **LoopedFlag**: 🔗

LoopedFlag **LOOPED_FLAG_NONE** = `0`

This flag indicates that the animation proceeds without any looping.

LoopedFlag **LOOPED_FLAG_END** = `1`

This flag indicates that the animation has reached the end of the animation and just after loop processed.

LoopedFlag **LOOPED_FLAG_START** = `2`

This flag indicates that the animation has reached the start of the animation and just after loop processed.

---

enum **FindMode**: 🔗

FindMode **FIND_MODE_NEAREST** = `0`

Finds the nearest time key.

FindMode **FIND_MODE_APPROX** = `1`

Finds only the key with approximating the time.

FindMode **FIND_MODE_EXACT** = `2`

Finds only the key with matching the time.

Property Descriptions

bool **capture_included** = `false`

- bool **is_capture_included**()

Returns `true` if the capture track is included. This is a cached readonly value for performance.

---

float **length** = `1.0`

- void **set_length**(value: float)
- float **get_length**()

The total length of the animation (in seconds).

Note: Length is not delimited by the last key, as this one may be before or after the end to ensure correct interpolation and looping.

LoopMode **loop_mode** = `0`

- void **set_loop_mode**(value: LoopMode)
- LoopMode **get_loop_mode**()

Determines the behavior of both ends of the animation timeline during animation playback. This is used for correct interpolation of animation cycles, and for hinting the player that it must restart the animation.

---

float **step** = `0.0333333`

- void **set_step**(value: float)
- float **get_step**()

The animation step value.

Method Descriptions

int **add_track**(type: TrackType, at_position: int = -1) 🔗

Adds a track to the Animation.

StringName **animation_track_get_key_animation**(track_idx: int, key_idx: int) const 🔗

Returns the animation name at the key identified by `key_idx`. The `track_idx` must be the index of an Animation Track.

int **animation_track_insert_key**(track_idx: int, time: float, animation: StringName) 🔗

Inserts a key with value `animation` at the given `time` (in seconds). The `track_idx` must be the index of an Animation Track.

void **animation_track_set_key_animation**(track_idx: int, key_idx: int, animation: StringName) 🔗

Sets the key identified by `key_idx` to value `animation`. The `track_idx` must be the index of an Animation Track.

float **audio_track_get_key_end_offset**(track_idx: int, key_idx: int) const 🔗

Returns the end offset of the key identified by `key_idx`. The `track_idx` must be the index of an Audio Track.

End offset is the number of seconds cut off at the ending of the audio stream.

---

float **audio_track_get_key_start_offset**(track_idx: int, key_idx: int) const

Returns the start offset of the key identified by `key_idx`. The `track_idx` must be the index of an Audio Track.

Start offset is the number of seconds cut off at the beginning of the audio stream.

Resource **audio_track_get_key_stream**(track_idx: int, key_idx: int) const

Returns the audio stream of the key identified by `key_idx`. The `track_idx` must be the index of an Audio Track.

---

int **audio_track_insert_key**(track_idx: int, time: float, stream: Resource, start_offset: float = 0, end_offset: float = 0)

Inserts an Audio Track key at the given `time` in seconds. The `track_idx` must be the index of an Audio Track.

`stream` is the AudioStream resource to play. `start_offset` is the number of seconds cut off at the beginning of the audio stream, while `end_offset` is at the ending.

---

bool **audio_track_is_use_blend**(track_idx: int) const

Returns `true` if the track at `track_idx` will be blended with other animations.

---

void **audio_track_set_key_end_offset**(track_idx: int, key_idx: int, offset: float)

Sets the end offset of the key identified by `key_idx` to value `offset`. The `track_idx` must be the index of an Audio Track.

---

void **audio_track_set_key_start_offset**(track_idx: int, key_idx: int, offset: float)

Sets the start offset of the key identified by `key_idx` to value `offset`. The `track_idx` must be the index of an Audio Track.

---

void **audio_track_set_key_stream**(track_idx: int, key_idx: int, stream: Resource)

Sets the stream of the key identified by `key_idx` to value `stream`. The `track_idx` must be the index of an Audio Track.

---

void **audio_track_set_use_blend**(track_idx: int, enable: bool)

Sets whether the track will be blended with other animations. If `true`, the audio playback volume changes depending on the blend value.

Vector2 **bezier_track_get_key_in_handle**(track_idx: int, key_idx: int) const

Returns the in handle of the key identified by `key_idx`. The `track_idx` must be the index of a Bezier Track.

Vector2 **bezier_track_get_key_out_handle**(track_idx: int, key_idx: int) const

Returns the out handle of the key identified by `key_idx`. The `track_idx` must be the index of a Bezier Track.

---

float **bezier_track_get_key_value**(track_idx: int, key_idx: int) const

Returns the value of the key identified by `key_idx`. The `track_idx` must be the index of a Bezier Track.

---

int **bezier_track_insert_key**(track_idx: int, time: float, value: float, in_handle: Vector2 = Vector2(0, 0), out_handle: Vector2 = Vector2(0, 0))

Inserts a Bezier Track key at the given `time` in seconds. The `track_idx` must be the index of a Bezier Track.

`in_handle` is the left-side weight of the added Bezier curve point, `out_handle` is the right-side one, while `value` is the actual value at this point.

---

float **bezier_track_interpolate**(track_idx: int, time: float) const

Returns the interpolated value at the given `time` (in seconds). The `track_idx` must be the index of a Bezier Track.

---

void **bezier_track_set_key_in_handle**(track_idx: int, key_idx: int, in_handle: Vector2, balanced_value_time_ratio: float = 1.0)

Sets the in handle of the key identified by `key_idx` to value `in_handle`. The `track_idx` must be the index of a Bezier Track.

---

void **bezier_track_set_key_out_handle**(track_idx: int, key_idx: int, out_handle: Vector2, balanced_value_time_ratio: float = 1.0)

Sets the out handle of the key identified by `key_idx` to value `out_handle`. The `track_idx` must be the index of a Bezier Track.

---

void **bezier_track_set_key_value**(track_idx: int, key_idx: int, value: float)

Sets the value of the key identified by `key_idx` to the given value. The `track_idx` must be the index of a Bezier Track.

---

int **blend_shape_track_insert_key**(track_idx: int, time: float, amount: float)

Inserts a key in a given blend shape track. Returns the key index.

---

float **blend_shape_track_interpolate**(track_idx: int, time_sec: float, backward: bool = false) const

Returns the interpolated blend shape value at the given time (in seconds). The `track_idx` must be the index of a blend shape track.

---

void **clear**()

Clear the animation (clear all tracks and reset all).

---

void **compress**(page_size: int = 8192, fps: int = 120, split_tolerance: float = 4.0) 🔗

Compress the animation and all its tracks in-place. This will make track_is_compressed return `true` once called on this **Animation**. Compressed tracks require less memory to be played, and are designed to be used for complex 3D animations (such as cutscenes) imported from external 3D software. Compression is lossy, but the difference is usually not noticeable in real world conditions.

Note: Compressed tracks have various limitations (such as not being editable from the editor), so only use compressed animations if you actually need them.

---

void **copy_track**(track_idx: int, to_animation: Animation) 🔗

Adds a new track to `to_animation` that is a copy of the given track from this animation.

---

int **find_track**(path: NodePath, type: TrackType) const 🔗

Returns the index of the specified track. If the track is not found, return -1.

---

int **get_track_count**() const 🔗

Returns the amount of tracks in the animation.

StringName **method_track_get_name**(track_idx: int, key_idx: int) const 🔗

Returns the method name of a method track.

Array **method_track_get_params**(track_idx: int, key_idx: int) const

Returns the arguments values to be called on a method track for a given key in a given track.

---

int **position_track_insert_key**(track_idx: int, time: float, position: Vector3)

Inserts a key in a given 3D position track. Returns the key index.

Vector3 **position_track_interpolate**(track_idx: int, time_sec: float, backward: bool = false) const

Returns the interpolated position value at the given time (in seconds). The `track_idx` must be the index of a 3D position track.

---

void **remove_track**(track_idx: int)

Removes a track by specifying the track index.

---

int **rotation_track_insert_key**(track_idx: int, time: float, rotation: Quaternion)

Inserts a key in a given 3D rotation track. Returns the key index.

Quaternion **rotation_track_interpolate**(track_idx: int, time_sec: float, backward: bool = false) const

Returns the interpolated rotation value at the given time (in seconds). The `track_idx` must be the index of a 3D rotation track.

int **scale_track_insert_key**(track_idx: int, time: float, scale: Vector3)

Inserts a key in a given 3D scale track. Returns the key index.

Vector3 **scale_track_interpolate**(track_idx: int, time_sec: float, backward: bool = false) const

Returns the interpolated scale value at the given time (in seconds). The `track_idx` must be the index of a 3D scale track.

int **track_find_key**(track_idx: int, time: float, find_mode: FindMode = 0, limit: bool = false, backward: bool = false) const

Finds the key index by time in a given track. Optionally, only find it if the approx/exact time is given.

If `limit` is `true`, it does not return keys outside the animation range.

If `backward` is `true`, the direction is reversed in methods that rely on one directional processing.

For example, in case `find_mode` is FIND_MODE_NEAREST, if there is no key in the current position just after seeked, the first key found is retrieved by searching before the position, but if `backward` is `true`, the first key found is retrieved after the position.

bool **track_get_interpolation_loop_wrap**(track_idx: int) const

Returns `true` if the track at `track_idx` wraps the interpolation loop. New tracks wrap the interpolation loop by default.

InterpolationType **track_get_interpolation_type**(track_idx: int) const

Returns the interpolation type of a given track.

---

int **track_get_key_count**(track_idx: int) const

Returns the number of keys in a given track.

---

float **track_get_key_time**(track_idx: int, key_idx: int) const

Returns the time at which the key is located.

---

float **track_get_key_transition**(track_idx: int, key_idx: int) const

Returns the transition curve (easing) for a specific key (see the built-in math function @GlobalScope.ease).

Variant **track_get_key_value**(track_idx: int, key_idx: int) const

Returns the value of a given key in a given track.

NodePath **track_get_path**(track_idx: int) const

Gets the path of a track. For more information on the path format, see track_set_path.

TrackType **track_get_type**(track_idx: int) const

Gets the type of a track.

---

int **track_insert_key**(track_idx: int, time: float, key: Variant, transition: float = 1) 🔗

Inserts a generic key in a given track. Returns the key index.

---

bool **track_is_compressed**(track_idx: int) const 🔗

Returns `true` if the track is compressed, `false` otherwise. See also compress.

---

bool **track_is_enabled**(track_idx: int) const 🔗

Returns `true` if the track at index `track_idx` is enabled.

---

bool **track_is_imported**(track_idx: int) const 🔗

Returns `true` if the given track is imported. Else, return `false`.

---

void **track_move_down**(track_idx: int) 🔗

Moves a track down.

---

void **track_move_to**(track_idx: int, to_idx: int) 🔗

Changes the index position of track `track_idx` to the one defined in `to_idx`.

---

void **track_move_up**(track_idx: int) 🔗

Moves a track up.

---

void **track_remove_key**(track_idx: int, key_idx: int)

Removes a key by index in a given track.

---

void **track_remove_key_at_time**(track_idx: int, time: float)

Removes a key at `time` in a given track.

---

void **track_set_enabled**(track_idx: int, enabled: bool)

Enables/disables the given track. Tracks are enabled by default.

---

void **track_set_imported**(track_idx: int, imported: bool)

Sets the given track as imported or not.

---

void **track_set_interpolation_loop_wrap**(track_idx: int, interpolation: bool)

If `true`, the track at `track_idx` wraps the interpolation loop.

---

void **track_set_interpolation_type**(track_idx: int, interpolation: InterpolationType)

Sets the interpolation type of a given track.

---

void **track_set_key_time**(track_idx: int, key_idx: int, time: float)

Sets the time of an existing key.

---

void **track_set_key_transition**(track_idx: int, key_idx: int, transition: float)

Sets the transition curve (easing) for a specific key (see the built-in math function @GlobalScope.ease).

---

void **track_set_key_value**(track_idx: int, key: int, value: Variant)

Sets the value of an existing key.

---

void **track_set_path**(track_idx: int, path: NodePath)

Sets the path of a track. Paths must be valid scene-tree paths to a node and must be specified starting from the AnimationMixer.root_node that will reproduce the animation. Tracks that control properties or bones must append their name after the path, separated by `":"`.

For example, `"character/skeleton:ankle"` or `"character/mesh:transform/local"`.

---

void **track_swap**(track_idx: int, with_idx: int)

Swaps the track `track_idx`'s index position with the track `with_idx`.

UpdateMode **value_track_get_update_mode**(track_idx: int) const

Returns the update mode of a value track.

Variant **value_track_interpolate**(track_idx: int, time_sec: float, backward: bool = false) const 🔗

Returns the interpolated value at the given time (in seconds). The `track_idx` must be the index of a value track.

A `backward` mainly affects the direction of key retrieval of the track with UPDATE_DISCRETE converted by AnimationMixer.ANIMATION_CALLBACK_MODE_DISCRETE_FORCE_CONTINUOUS to match the result with track_find_key.

void **value_track_set_update_mode**(track_idx: int, mode: UpdateMode) 🔗

Sets the update mode (see UpdateMode) of a value track.

AnimationLibrary

Inherits: Resource **<** RefCounted **<** Object

Container for Animation resources.

Description

An animation library stores a set of animations accessible through StringName keys, for use with AnimationPlayer nodes.

Tutorials

- Animation tutorial index

Methods

Error	add_animation(name: StringName, animation: Animation)
Animation	get_animation(name: StringName) const
Array[StringName]	get_animation_list() const
bool	has_animation(name: StringName) const

void	remove_animation(name: StringName)
void	rename_animation(name: StringName, newname: StringName)

Signals

animation_added(name: StringName)

Emitted when an Animation is added, under the key `name`.

animation_changed(name: StringName)

Emitted when there's a change in one of the animations, e.g. tracks are added, moved or have changed paths. `name` is the key of the animation that was changed.

See also Resource.changed, which this acts as a relay for.

animation_removed(name: StringName)

Emitted when an Animation stored with the key `name` is removed.

animation_renamed(name: StringName, to_name: StringName)

Emitted when the key for an Animation is changed, from `name` to `to_name`.

Method Descriptions

Error **add_animation**(name: StringName, animation: Animation)

Adds the `animation` to the library, accessible by the key `name`.

Animation **get_animation**(name: StringName) const

Returns the Animation with the key `name`. If the animation does not exist, `null` is returned and an error is logged.

Array[StringName] **get_animation_list**() const

Returns the keys for the Animations stored in the library.

---

bool **has_animation**(name: StringName) const

Returns `true` if the library stores an Animation with `name` as the key.

---

void **remove_animation**(name: StringName)

Removes the Animation with the key `name`.

---

void **rename_animation**(name: StringName, newname: StringName)

Changes the key of the Animation associated with the key `name` to `newname`.

AnimationNode

Inherits: Resource **<** RefCounted **<** Object

Inherited By: AnimationNodeOutput, AnimationNodeSync, AnimationNodeTimeScale, AnimationNodeTimeSeek, AnimationRootNode

Base class for AnimationTree nodes. Not related to scene nodes.

Description

Base resource for AnimationTree nodes. In general, it's not used directly, but you can create custom ones with custom blending formulas.

Inherit this when creating animation nodes mainly for use in AnimationNodeBlendTree, otherwise AnimationRootNode should be used instead.

You can access the time information as read-only parameter which is processed and stored in the previous frame for all nodes except AnimationNodeOutput.

Note: If multiple inputs exist in the **AnimationNode**, which time information takes precedence depends on the type of **AnimationNode**.

```
var current_length =
$AnimationTree[parameters/AnimationNodeName/current_length]
var current_position =
$AnimationTree[parameters/AnimationNodeName/current_position]
var current_delta =
$AnimationTree[parameters/AnimationNodeName/current_delta]
```

Tutorials

- Using AnimationTree

Properties

bool	filter_enabled

Methods

String	_get_caption() virtual const
AnimationNode	_get_child_by_name(name: StringName) virtual const
Dictionary	_get_child_nodes() virtual const
Variant	_get_parameter_default_value(parameter: StringName) virtual const
Array	_get_parameter_list() virtual const
bool	_has_filter() virtual const

bool	_is_parameter_read_only(parameter: StringName) virtual const
float	_process(time: float, seek: bool, is_external_seeking: bool, test_only: bool) virtual const
bool	add_input(name: String)
void	blend_animation(animation: StringName, time: float, delta: float, seeked: bool, is_external_seeking: bool, blend: float, looped_flag: LoopedFlag = 0)
float	blend_input(input_index: int, time: float, seek: bool, is_external_seeking: bool, blend: float, filter: FilterAction = 0, sync: bool = true, test_only: bool = false)
float	blend_node(name: StringName, node: AnimationNode, time: float, seek: bool, is_external_seeking: bool, blend: float, filter: FilterAction = 0, sync: bool = true, test_only: bool = false)
int	find_input(name: String) const
int	get_input_count() const

String	get_input_name(input: int) const
Variant	get_parameter(name: StringName) const
bool	is_path_filtered(path: NodePath) const
void	remove_input(index: int)
void	set_filter_path(path: NodePath, enable: bool)
bool	set_input_name(input: int, name: String)
void	set_parameter(name: StringName, value: Variant)

Signals

animation_node_removed(object_id: int, name: String) 🔗

Emitted by nodes that inherit from this class and that have an internal tree when one of their animation nodes removes. The animation nodes that emit this signal are AnimationNodeBlendSpace1D, AnimationNodeBlendSpace2D, AnimationNodeStateMachine, and AnimationNodeBlendTree.

---

animation_node_renamed(object_id: int, old_name: String, new_name: String) 🔗

Emitted by nodes that inherit from this class and that have an internal tree when one of their animation node names changes. The animation nodes that emit this signal are AnimationNodeBlendSpace1D, AnimationNodeBlendSpace2D, AnimationNodeStateMachine, and AnimationNodeBlendTree.

---

tree_changed() 🔗

Emitted by nodes that inherit from this class and that have an internal tree when one of their animation nodes changes. The animation nodes that emit this signal are AnimationNodeBlendSpace1D, AnimationNodeBlendSpace2D, AnimationNodeStateMachine, AnimationNodeBlendTree and AnimationNodeTransition.

Enumerations

enum **FilterAction**: 🔗

FilterAction **FILTER_IGNORE** = `0`

Do not use filtering.

FilterAction **FILTER_PASS** = `1`

Paths matching the filter will be allowed to pass.

FilterAction **FILTER_STOP** = `2`

Paths matching the filter will be discarded.

FilterAction **FILTER_BLEND** = `3`

Paths matching the filter will be blended (by the blend value).

Property Descriptions

bool **filter_enabled**

- void **set_filter_enabled**(value: bool)
- bool **is_filter_enabled**()

If `true`, filtering is enabled.

Method Descriptions

String **_get_caption**() virtual const

When inheriting from AnimationRootNode, implement this virtual method to override the text caption for this animation node.

AnimationNode **_get_child_by_name**(name: StringName) virtual const

When inheriting from AnimationRootNode, implement this virtual method to return a child animation node by its `name`.

Dictionary **_get_child_nodes**() virtual const

When inheriting from AnimationRootNode, implement this virtual method to return all child animation nodes in order as a `name: node` dictionary.

Variant **_get_parameter_default_value**(parameter: StringName) virtual const 🔗

When inheriting from AnimationRootNode, implement this virtual method to return the default value of a `parameter`. Parameters are custom local memory used for your animation nodes, given a resource can be reused in multiple trees.

Array **_get_parameter_list**() virtual const 🔗

When inheriting from AnimationRootNode, implement this virtual method to return a list of the properties on this animation node. Parameters are custom local memory used for your animation nodes, given a resource can be reused in multiple trees. Format is similar to Object.get_property_list.

---

bool **_has_filter**() virtual const 🔗

When inheriting from AnimationRootNode, implement this virtual method to return whether the blend tree editor should display filter editing on this animation node.

---

bool **_is_parameter_read_only**(parameter: StringName) virtual const 🔗

When inheriting from AnimationRootNode, implement this virtual method to return whether the `parameter` is read-only. Parameters are custom local memory used for your animation nodes, given a resource can be reused in multiple trees.

---

float **_process**(time: float, seek: bool, is_external_seeking: bool, test_only: bool) virtual const 🔗

Deprecated: Currently this is mostly useless as there is a lack of many APIs to extend AnimationNode by GDScript. It is planned that a more flexible API using structures will be provided in the future.

When inheriting from AnimationRootNode, implement this virtual method to run some code when this animation node is processed. The `time` parameter is a relative delta, unless `seek` is `true`, in which case it is absolute.

Here, call the blend_input, blend_node or blend_animation functions. You can also use get_parameter and set_parameter to modify local memory.

This function should return the delta.

---

bool **add_input**(name: String) 🔗

Adds an input to the animation node. This is only useful for animation nodes created for use in an AnimationNodeBlendTree. If the addition fails, returns `false`.

---

void **blend_animation**(animation: StringName, time: float, delta: float, seeked: bool, is_external_seeking: bool, blend: float, looped_flag: LoopedFlag = 0) 🔗

Blend an animation by `blend` amount (name must be valid in the linked AnimationPlayer). A `time` and `delta` may be passed, as well as whether `seeked` happened.

A `looped_flag` is used by internal processing immediately after the loop. See also LoopedFlag.

---

float **blend_input**(input_index: int, time: float, seek: bool, is_external_seeking: bool, blend: float, filter: FilterAction = 0, sync:

bool = true, test_only: bool = false) 🔗

Blend an input. This is only useful for animation nodes created for an AnimationNodeBlendTree. The `time` parameter is a relative delta, unless `seek` is `true`, in which case it is absolute. A filter mode may be optionally passed (see FilterAction for options).

---

float **blend_node**(name: StringName, node: AnimationNode, time: float, seek: bool, is_external_seeking: bool, blend: float, filter: FilterAction = 0, sync: bool = true, test_only: bool = false) 🔗

Blend another animation node (in case this animation node contains child animation nodes). This function is only useful if you inherit from AnimationRootNode instead, otherwise editors will not display your animation node for addition.

---

int **find_input**(name: String) const 🔗

Returns the input index which corresponds to `name`. If not found, returns `-1`.

---

int **get_input_count**() const 🔗

Amount of inputs in this animation node, only useful for animation nodes that go into AnimationNodeBlendTree.

String **get_input_name**(input: int) const 🔗

Gets the name of an input by index.

Variant **get_parameter**(name: StringName) const 🔗

Gets the value of a parameter. Parameters are custom local memory used for your animation nodes, given a resource can be reused in multiple trees.

bool **is_path_filtered**(path: NodePath) const

Returns whether the given path is filtered.

void **remove_input**(index: int)

Removes an input, call this only when inactive.

void **set_filter_path**(path: NodePath, enable: bool)

Adds or removes a path for the filter.

bool **set_input_name**(input: int, name: String)

Sets the name of the input at the given `input` index. If the setting fails, returns `false`.

void **set_parameter**(name: StringName, value: Variant)

Sets a custom parameter. These are used as local memory, because resources can be reused across the tree or scenes.

AnimationNodeAdd2

Inherits: AnimationNodeSync **<** AnimationNode **<** Resource **<** RefCounted **<** Object

Blends two animations additively inside of an AnimationNodeBlendTree.

Description

A resource to add to an AnimationNodeBlendTree. Blends two animations additively based on the amount value.

If the amount is greater than `1.0`, the animation connected to "in" port is blended with the amplified animation connected to "add" port.

If the amount is less than `0.0`, the animation connected to "in" port is blended with the inverted animation connected to "add" port.

Tutorials

- Using AnimationTree

AnimationNodeAdd3

Inherits: AnimationNodeSync **<** AnimationNode **<** Resource **<** RefCounted **<** Object

Blends two of three animations additively inside of an AnimationNodeBlendTree.

Description

A resource to add to an AnimationNodeBlendTree. Blends two animations out of three additively out of three based on the amount value.

This animation node has three inputs:

- The base animation to add to
- A "-add" animation to blend with when the blend amount is negative
- A "+add" animation to blend with when the blend amount is positive

If the absolute value of the amount is greater than `1.0`, the animation connected to "in" port is blended with the amplified animation connected to "-add"/"+add" port.

Tutorials

- Using AnimationTree
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

AnimationNodeAnimation

Inherits: AnimationRootNode **<** AnimationNode **<** Resource **<** RefCounted **<** Object

An input animation for an AnimationNodeBlendTree.

Description

A resource to add to an AnimationNodeBlendTree. Only has one output port using the animation property. Used as an input for AnimationNodes that blend animations together.

Tutorials

- Using AnimationTree
- 3D Platformer Demo [https://godotengine.org/asset-library/asset/2748]
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

StringName	animation	`&""`
LoopMode	loop_mode	
PlayMode	play_mode	`0`

float	start_offset	
bool	stretch_time_scale	
float	timeline_length	
bool	use_custom_timeline	`false`

Enumerations

enum **PlayMode**: 🔗

PlayMode **PLAY_MODE_FORWARD** = `0`

Plays animation in forward direction.

PlayMode **PLAY_MODE_BACKWARD** = `1`

Plays animation in backward direction.

Property Descriptions

StringName **animation** = `&""` 🔗

- void **set_animation**(value: StringName)
- StringName **get_animation**()

Animation to use as an output. It is one of the animations provided by AnimationTree.anim_player.

LoopMode **loop_mode** 🔗

- void **set_loop_mode**(value: LoopMode)
- LoopMode **get_loop_mode**()

If use_custom_timeline is `true`, override the loop settings of the original Animation resource with the value.

Note: If the Animation.loop_mode isn't set to looping, the Animation.track_set_interpolation_loop_wrap option will not be respected. If you cannot get the expected behavior, consider duplicating the Animation resource and changing the loop settings.

PlayMode **play_mode** = `0` 🔗

- void **set_play_mode**(value: PlayMode)
- PlayMode **get_play_mode**()

Determines the playback direction of the animation.

---

float **start_offset** 🔗

- void **set_start_offset**(value: float)
- float **get_start_offset**()

If use_custom_timeline is `true`, offset the start position of the animation.

This is useful for adjusting which foot steps first in 3D walking animations.

---

bool **stretch_time_scale**

- void **set_stretch_time_scale**(value: bool)
- bool **is_stretching_time_scale**()

If `true`, scales the time so that the length specified in timeline_length is one cycle.

This is useful for matching the periods of walking and running animations.

If `false`, the original animation length is respected. If you set the loop to loop_mode, the animation will loop in timeline_length.

---

float **timeline_length**

- void **set_timeline_length**(value: float)
- float **get_timeline_length**()

If use_custom_timeline is `true`, offset the start position of the animation.

---

bool **use_custom_timeline** = `false`

- void **set_use_custom_timeline**(value: bool)
- bool **is_using_custom_timeline**()

If `true`, AnimationNode provides an animation based on the Animation resource with some parameters adjusted.

AnimationNodeBlend2

Inherits: AnimationNodeSync **<** AnimationNode **<** Resource **<** RefCounted **<** Object

Blends two animations linearly inside of an AnimationNodeBlendTree.

Description

A resource to add to an AnimationNodeBlendTree. Blends two animations linearly based on the amount value.

In general, the blend value should be in the `[0.0, 1.0]` range. Values outside of this range can blend amplified or inverted animations, however, AnimationNodeAdd2 works better for this purpose.

Tutorials

- Using AnimationTree
- 3D Platformer Demo [https://godotengine.org/asset-library/asset/2748]
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

AnimationNodeBlend3

Inherits: AnimationNodeSync **<** AnimationNode **<** Resource **<** RefCounted **<** Object

Blends two of three animations linearly inside of an AnimationNodeBlendTree.

Description

A resource to add to an AnimationNodeBlendTree. Blends two animations out of three linearly out of three based on the amount value.

This animation node has three inputs:

- The base animation to blend with
- A "-blend" animation to blend with when the blend amount is negative value
- A "+blend" animation to blend with when the blend amount is positive value

In general, the blend value should be in the `[-1.0, 1.0]` range. Values outside of this range can blend amplified animations, however, AnimationNodeAdd3 works better for this purpose.

Tutorials

- Using AnimationTree

AnimationNodeBlendSpace1D

Inherits: AnimationRootNode **<** AnimationNode **<** Resource **<** RefCounted **<** Object

A set of AnimationRootNodes placed on a virtual axis, crossfading between the two adjacent ones. Used by AnimationTree.

Description

A resource used by AnimationNodeBlendTree.

AnimationNodeBlendSpace1D represents a virtual axis on which any type of AnimationRootNodes can be added using add_blend_point. Outputs the linear blend of the two AnimationRootNodes adjacent to the current value.

You can set the extents of the axis with min_space and max_space.

Tutorials

- Using AnimationTree

Properties

BlendMode	blend_mode	`0`
float	max_space	`1.0`

float	min_space	`-1.0`
float	snap	`0.1`
bool	sync	`false`
String	value_label	`"value"`

Methods

void	add_blend_point(node: AnimationRootNode, pos: float, at_index: int = -1)
int	get_blend_point_count() const
AnimationRootNode	get_blend_point_node(point: int) const
float	get_blend_point_position(point: int) const
void	remove_blend_point(point: int)
void	set_blend_point_node(point: int, node: AnimationRootNode)

void	set_blend_point_position(point: int, pos: float)

Enumerations

enum **BlendMode**: 🔗

BlendMode **BLEND_MODE_INTERPOLATED** = `0`

The interpolation between animations is linear.

BlendMode **BLEND_MODE_DISCRETE** = `1`

The blend space plays the animation of the animation node which blending position is closest to. Useful for frame-by-frame 2D animations.

BlendMode **BLEND_MODE_DISCRETE_CARRY** = `2`

Similar to BLEND_MODE_DISCRETE, but starts the new animation at the last animation's playback position.

Property Descriptions

BlendMode **blend_mode** = `0` 🔗

- void **set_blend_mode**(value: BlendMode)
- BlendMode **get_blend_mode**()

Controls the interpolation between animations. See BlendMode constants.

---

float **max_space** = `1.0`

- void **set_max_space**(value: float)
- float **get_max_space**()

The blend space's axis's upper limit for the points' position. See add_blend_point.

---

float **min_space** = `-1.0`

- void **set_min_space**(value: float)
- float **get_min_space**()

The blend space's axis's lower limit for the points' position. See add_blend_point.

---

float **snap** = `0.1`

- void **set_snap**(value: float)
- float **get_snap**()

Position increment to snap to when moving a point on the axis.

---

bool **sync** = `false`

- void **set_use_sync**(value: bool)
- bool **is_using_sync**()

If `false`, the blended animations' frame are stopped when the blend value is `0`.

If `true`, forcing the blended animations to advance frame.

String **value_label** = `"value"` 🔗

- void **set_value_label**(value: String)
- String **get_value_label**()

Label of the virtual axis of the blend space.

Method Descriptions

void **add_blend_point**(node: AnimationRootNode, pos: float, at_index: int = -1) 🔗

Adds a new point that represents a `node` on the virtual axis at a given position set by `pos`. You can insert it at a specific index using the `at_index` argument. If you use the default value for `at_index`, the point is inserted at the end of the blend points array.

---

int **get_blend_point_count**() const 🔗

Returns the number of points on the blend axis.

AnimationRootNode **get_blend_point_node**(point: int) const 🔗

Returns the AnimationNode referenced by the point at index `point`.

---

float **get_blend_point_position**(point: int) const 🔗

Returns the position of the point at index `point`.

---

void **remove_blend_point**(point: int) 🔗

Removes the point at index `point` from the blend axis.

void **set_blend_point_node**(point: int, node: AnimationRootNode)

Changes the AnimationNode referenced by the point at index `point`.

void **set_blend_point_position**(point: int, pos: float)

Updates the position of the point at index `point` on the blend axis.

AnimationNodeBlendSpace2D

Inherits: AnimationRootNode **<** AnimationNode **<** Resource **<** RefCounted **<** Object

A set of AnimationRootNodes placed on 2D coordinates, crossfading between the three adjacent ones. Used by AnimationTree.

Description

A resource used by AnimationNodeBlendTree.

AnimationNodeBlendSpace2D represents a virtual 2D space on which AnimationRootNodes are placed. Outputs the linear blend of the three adjacent animations using a Vector2 weight. Adjacent in this context means the three AnimationRootNodes making up the triangle that contains the current value.

You can add vertices to the blend space with add_blend_point and automatically triangulate it by setting auto_triangles to `true`. Otherwise, use add_triangle and remove_triangle to triangulate the blend space by hand.

Tutorials

- Using AnimationTree
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

bool	auto_triangles	`true`
BlendMode	blend_mode	`0`
Vector2	max_space	`Vector2(1, 1)`
Vector2	min_space	`Vector2(-1, -1)`
Vector2	snap	`Vector2(0.1, 0.1)`
bool	sync	`false`
String	x_label	`"x"`
String	y_label	`"y"`

Methods

void	add_blend_point(node: AnimationRootNode, pos: Vector2, at_index: int = -1)
void	add_triangle(x: int, y: int, z: int, at_index: int = -1)

int	get_blend_point_count() const
AnimationRootNode	get_blend_point_node(point: int) const
Vector2	get_blend_point_position(point: int) const
int	get_triangle_count() const
int	get_triangle_point(triangle: int, point: int)
void	remove_blend_point(point: int)
void	remove_triangle(triangle: int)
void	set_blend_point_node(point: int, node: AnimationRootNode)
void	set_blend_point_position(point: int, pos: Vector2)

Signals

triangles_updated() 🔗

Emitted every time the blend space's triangles are created, removed, or when one of their vertices changes position.

Enumerations

enum **BlendMode**:

BlendMode **BLEND_MODE_INTERPOLATED** = `0`

The interpolation between animations is linear.

BlendMode **BLEND_MODE_DISCRETE** = `1`

The blend space plays the animation of the animation node which blending position is closest to. Useful for frame-by-frame 2D animations.

BlendMode **BLEND_MODE_DISCRETE_CARRY** = `2`

Similar to BLEND_MODE_DISCRETE, but starts the new animation at the last animation's playback position.

Property Descriptions

bool **auto_triangles** = `true`

- void **set_auto_triangles**(value: bool)
- bool **get_auto_triangles**()

If `true`, the blend space is triangulated automatically. The mesh updates every time you add or remove points with add_blend_point and remove_blend_point.

BlendMode **blend_mode** = `0`

- void **set_blend_mode**(value: BlendMode)
- BlendMode **get_blend_mode**()

Controls the interpolation between animations. See BlendMode constants.

Vector2 **max_space** = `Vector2(1, 1)`

- void **set_max_space**(value: Vector2)
- Vector2 **get_max_space**()

The blend space's X and Y axes' upper limit for the points' position. See add_blend_point.

Vector2 **min_space** = `Vector2(-1, -1)`

- void **set_min_space**(value: Vector2)
- Vector2 **get_min_space**()

The blend space's X and Y axes' lower limit for the points' position. See add_blend_point.

Vector2 **snap** = `Vector2(0.1, 0.1)`

- void **set_snap**(value: Vector2)
- Vector2 **get_snap**()

Position increment to snap to when moving a point.

---

bool **sync** = `false`

- void **set_use_sync**(value: bool)
- bool **is_using_sync**()

If `false`, the blended animations' frame are stopped when the blend value is `0`.

If `true`, forcing the blended animations to advance frame.

String **x_label** = `"x"`

- void **set_x_label**(value: String)
- String **get_x_label**()

Name of the blend space's X axis.

String **y_label** = `"y"`

- void **set_y_label**(value: String)
- String **get_y_label**()

Name of the blend space's Y axis.

Method Descriptions

void **add_blend_point**(node: AnimationRootNode, pos: Vector2, at_index: int = -1)

Adds a new point that represents a `node` at the position set by `pos`. You can insert it at a specific index using the `at_index` argument. If you use the default value for `at_index`, the point is inserted at the end of the blend points array.

---

void **add_triangle**(x: int, y: int, z: int, at_index: int = -1)

Creates a new triangle using three points `x`, `y`, and `z`. Triangles can overlap. You can insert the triangle at a specific index using the `at_index` argument. If you use the default value for `at_index`, the point is inserted at the end of the blend points array.

---

int **get_blend_point_count**() const 🔗

Returns the number of points in the blend space.

AnimationRootNode **get_blend_point_node**(point: int) const 🔗

Returns the AnimationRootNode referenced by the point at index `point`.

Vector2 **get_blend_point_position**(point: int) const 🔗

Returns the position of the point at index `point`.

---

int **get_triangle_count**() const 🔗

Returns the number of triangles in the blend space.

---

int **get_triangle_point**(triangle: int, point: int) 🔗

Returns the position of the point at index `point` in the triangle of index `triangle`.

---

void **remove_blend_point**(point: int) 🔗

Removes the point at index `point` from the blend space.

---

void **remove_triangle**(triangle: int)

Removes the triangle at index `triangle` from the blend space.

---

void **set_blend_point_node**(point: int, node: AnimationRootNode)

Changes the AnimationNode referenced by the point at index `point`.

---

void **set_blend_point_position**(point: int, pos: Vector2)

Updates the position of the point at index `point` in the blend space.

AnimationNodeBlendTree

Inherits: AnimationRootNode **<** AnimationNode **<** Resource **<** RefCounted **<** Object

A sub-tree of many type AnimationNodes used for complex animations. Used by AnimationTree.

Description

This animation node may contain a sub-tree of any other type animation nodes, such as AnimationNodeTransition, AnimationNodeBlend2, AnimationNodeBlend3, AnimationNodeOneShot, etc. This is one of the most commonly used animation node roots.

An AnimationNodeOutput node named `output` is created by default.

Tutorials

- Using AnimationTree

Properties

Vector2	graph_offset	`Vector2(0, 0)`

Methods

void	add_node(name: StringName, node: AnimationNode, position: Vector2 = Vector2(0, 0))
void	connect_node(input_node: StringName, input_index: int, output_node: StringName)
void	disconnect_node(input_node: StringName, input_index: int)
AnimationNode	get_node(name: StringName) const
Vector2	get_node_position(name: StringName) const
bool	has_node(name: StringName) const
void	remove_node(name: StringName)
void	rename_node(name: StringName, new_name: StringName)
void	set_node_position(name: StringName, position: Vector2)

Signals

node_changed(node_name: StringName) 🔗

Emitted when the input port information is changed.

Constants

CONNECTION_OK = `0`

The connection was successful.

CONNECTION_ERROR_NO_INPUT = `1`

The input node is `null`.

CONNECTION_ERROR_NO_INPUT_INDEX = `2`

The specified input port is out of range.

CONNECTION_ERROR_NO_OUTPUT = `3`

The output node is `null`.

CONNECTION_ERROR_SAME_NODE = `4`

Input and output nodes are the same.

CONNECTION_ERROR_CONNECTION_EXISTS = `5`

The specified connection already exists.

Property Descriptions

Vector2 **graph_offset** = `Vector2(0, 0)`

- void **set_graph_offset**(value: Vector2)

- Vector2 **get_graph_offset**()

The global offset of all sub animation nodes.

Method Descriptions

void **add_node**(name: StringName, node: AnimationNode, position: Vector2 = Vector2(0, 0)) 🔗

Adds an AnimationNode at the given `position`. The `name` is used to identify the created sub animation node later.

---

void **connect_node**(input_node: StringName, input_index: int, output_node: StringName) 🔗

Connects the output of an AnimationNode as input for another AnimationNode, at the input port specified by `input_index`.

---

void **disconnect_node**(input_node: StringName, input_index: int) 🔗

Disconnects the animation node connected to the specified input.

AnimationNode **get_node**(name: StringName) const 🔗

Returns the sub animation node with the specified `name`.

Vector2 **get_node_position**(name: StringName) const 🔗

Returns the position of the sub animation node with the specified `name`.

bool **has_node**(name: StringName) const

Returns `true` if a sub animation node with specified `name` exists.

void **remove_node**(name: StringName)

Removes a sub animation node.

void **rename_node**(name: StringName, new_name: StringName)

Changes the name of a sub animation node.

void **set_node_position**(name: StringName, position: Vector2)

Modifies the position of a sub animation node.

AnimationNodeOneShot

Inherits: AnimationNodeSync **<** AnimationNode **<** Resource **<** RefCounted **<** Object

Plays an animation once in an AnimationNodeBlendTree.

Description

A resource to add to an AnimationNodeBlendTree. This animation node will execute a sub-animation and return once it finishes. Blend times for fading in and out can be customized, as well as filters.

After setting the request and changing the animation playback, the one-shot node automatically clears the request on the next process frame by setting its `request` value to ONE_SHOT_REQUEST_NONE.

GDScript

```
# Play child animation connected to "shot" port.
animation_tree.set("parameters/OneShot/request",
AnimationNodeOneShot.ONE_SHOT_REQUEST_FIRE)
# Alternative syntax (same result as above).
animation_tree["parameters/OneShot/request"] =
AnimationNodeOneShot.ONE_SHOT_REQUEST_FIRE

# Abort child animation connected to "shot" port.
animation_tree.set("parameters/OneShot/request",
AnimationNodeOneShot.ONE_SHOT_REQUEST_ABORT)
# Alternative syntax (same result as above).
animation_tree["parameters/OneShot/request"] =
AnimationNodeOneShot.ONE_SHOT_REQUEST_ABORT

# Abort child animation with fading out connected to "shot"
port.
animation_tree.set("parameters/OneShot/request",
AnimationNodeOneShot.ONE_SHOT_REQUEST_FADE_OUT)
# Alternative syntax (same result as above).
animation_tree["parameters/OneShot/request"] =
AnimationNodeOneShot.ONE_SHOT_REQUEST_FADE_OUT

# Get current state (read-only).
animation_tree.get("parameters/OneShot/active")
# Alternative syntax (same result as above).
animation_tree["parameters/OneShot/active"]

# Get current internal state (read-only).
animation_tree.get("parameters/OneShot/internal_active")
# Alternative syntax (same result as above).
animation_tree["parameters/OneShot/internal_active"]
```

C#

```
// Play child animation connected to "shot" port.
animationTree.Set("parameters/OneShot/request",
(int)AnimationNodeOneShot.OneShotRequest.Fire);

// Abort child animation connected to "shot" port.
animationTree.Set("parameters/OneShot/request",
(int)AnimationNodeOneShot.OneShotRequest.Abort);

// Abort child animation with fading out connected to "shot"
port.
animationTree.Set("parameters/OneShot/request",
(int)AnimationNodeOneShot.OneShotRequest.FadeOut);

// Get current state (read-only).
animationTree.Get("parameters/OneShot/active");

// Get current internal state (read-only).
animationTree.Get("parameters/OneShot/internal_active");
```

Tutorials

- Using AnimationTree

- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

bool	autorestart	`false`
float	autorestart_delay	`1.0`
float	autorestart_random_delay	`0.0`
bool	break_loop_at_end	`false`
Curve	fadein_curve	
float	fadein_time	`0.0`
Curve	fadeout_curve	
float	fadeout_time	`0.0`
MixMode	mix_mode	`0`

Enumerations

enum **OneShotRequest**:

OneShotRequest **ONE_SHOT_REQUEST_NONE** = `0`

The default state of the request. Nothing is done.

OneShotRequest **ONE_SHOT_REQUEST_FIRE** = `1`

The request to play the animation connected to "shot" port.

OneShotRequest **ONE_SHOT_REQUEST_ABORT** = `2`

The request to stop the animation connected to "shot" port.

OneShotRequest **ONE_SHOT_REQUEST_FADE_OUT** = `3`

The request to fade out the animation connected to "shot" port.

enum **MixMode**:

MixMode **MIX_MODE_BLEND** = `0`

Blends two animations. See also AnimationNodeBlend2.

MixMode **MIX_MODE_ADD** = `1`

Blends two animations additively. See also AnimationNodeAdd2.

Property Descriptions

bool **autorestart** = `false`

- void **set_autorestart**(value: bool)
- bool **has_autorestart**()

If `true`, the sub-animation will restart automatically after finishing.

In other words, to start auto restarting, the animation must be played once with the ONE_SHOT_REQUEST_FIRE request. The ONE_SHOT_REQUEST_ABORT request stops the auto restarting, but it does not disable the autorestart itself. So, the ONE_SHOT_REQUEST_FIRE request will start auto restarting again.

---

float **autorestart_delay** = `1.0` 🔗

- void **set_autorestart_delay**(value: float)
- float **get_autorestart_delay**()

The delay after which the automatic restart is triggered, in seconds.

---

float **autorestart_random_delay** = `0.0` 🔗

- void **set_autorestart_random_delay**(value: float)
- float **get_autorestart_random_delay**()

If autorestart is `true`, a random additional delay (in seconds) between 0 and this value will be added to autorestart_delay.

---

bool **break_loop_at_end** = `false` 🔗

- void **set_break_loop_at_end**(value: bool)
- bool **is_loop_broken_at_end**()

If `true`, breaks the loop at the end of the loop cycle for transition, even if the animation is looping.

Curve **fadein_curve** 🔗

- void **set_fadein_curve**(value: Curve)
- Curve **get_fadein_curve**()

Determines how cross-fading between animations is eased. If empty, the transition will be linear.

float **fadein_time** = `0.0`

- void **set_fadein_time**(value: float)
- float **get_fadein_time**()

The fade-in duration. For example, setting this to `1.0` for a 5 second length animation will produce a cross-fade that starts at 0 second and ends at 1 second during the animation.

Note: AnimationNodeOneShot transitions the current state after the end of the fading. When AnimationNodeOutput is considered as the most upstream, so the fadein_time is scaled depending on the downstream delta. For example, if this value is set to `1.0` and a AnimationNodeTimeScale with a value of `2.0` is chained downstream, the actual processing time will be 0.5 second.

Curve **fadeout_curve**

- void **set_fadeout_curve**(value: Curve)
- Curve **get_fadeout_curve**()

Determines how cross-fading between animations is eased. If empty, the transition will be linear.

float **fadeout_time** = `0.0`

- void **set_fadeout_time**(value: float)
- float **get_fadeout_time**()

The fade-out duration. For example, setting this to `1.0` for a 5 second length animation will produce a cross-fade that starts at 4 second and ends at 5 second during the animation.

Note: AnimationNodeOneShot transitions the current state after the end of the fading. When AnimationNodeOutput is considered as the most upstream, so the fadeout_time is scaled depending on the downstream delta. For example, if this value is set to `1.0` and an AnimationNodeTimeScale with a value of `2.0` is chained downstream, the actual processing time will be 0.5 second.

MixMode **mix_mode** = `0`

- void **set_mix_mode**(value: MixMode)
- MixMode **get_mix_mode**()

The blend type.

AnimationNodeOutput

Inherits: AnimationNode **<** Resource **<** RefCounted **<** Object

The animation output node of an AnimationNodeBlendTree.

Description

A node created automatically in an AnimationNodeBlendTree that outputs the final animation.

Tutorials

- Using AnimationTree
- 3D Platformer Demo [https://godotengine.org/asset-library/asset/2748]
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

AnimationNodeStateMachine

Inherits: AnimationRootNode **<** AnimationNode **<** Resource **<** RefCounted **<** Object

A state machine with multiple AnimationRootNodes, used by AnimationTree.

Description

Contains multiple AnimationRootNodes representing animation states, connected in a graph. State transitions can be configured to happen automatically or via code, using a shortest-path algorithm. Retrieve the AnimationNodeStateMachinePlayback object from the AnimationTree node to control it programmatically.

Example:

GDScript

```
var state_machine = $AnimationTree.get("parameters/playback")
state_machine.travel("some_state")
```

C#

```
var stateMachine = GetNode<AnimationTree>
("AnimationTree").Get("parameters/playback") as
AnimationNodeStateMachinePlayback;
stateMachine.Travel("some_state");
```

Tutorials

- Using AnimationTree

Properties

bool	allow_transition_to_self	`false`

bool	reset_ends	`false`
StateMachineType	state_machine_type	`0`

Methods

void	add_node(name: StringName, node: AnimationNode, position: Vector2 = Vector2(0, 0))
void	add_transition(from: StringName, to: StringName, transition: AnimationNodeStateMachineTransition)
Vector2	get_graph_offset() const
AnimationNode	get_node(name: StringName) const
StringName	get_node_name(node: AnimationNode) const
Vector2	get_node_position(name: StringName) const
AnimationNodeStateMachineTransition	get_transition(idx: int) const
int	get_transition_count() const
StringName	get_transition_from(idx: int) const

StringName	get_transition_to(idx: int) const
bool	has_node(name: StringName) const
bool	has_transition(from: StringName, to: StringName) const
void	remove_node(name: StringName)
void	remove_transition(from: StringName, to: StringName)
void	remove_transition_by_index(idx: int)
void	rename_node(name: StringName, new_name: StringName)
void	replace_node(name: StringName, node: AnimationNode)
void	set_graph_offset(offset: Vector2)
void	set_node_position(name: StringName, position: Vector2)

Enumerations

enum **StateMachineType**: 🔗

StateMachineType **STATE_MACHINE_TYPE_ROOT** = `0`

Seeking to the beginning is treated as playing from the start state. Transition to the end state is treated as exiting the state machine.

StateMachineType **STATE_MACHINE_TYPE_NESTED** = `1`

Seeking to the beginning is treated as seeking to the beginning of the animation in the current state. Transition to the end state, or the absence of transitions in each state, is treated as exiting the state machine.

StateMachineType **STATE_MACHINE_TYPE_GROUPED** = `2`

This is a grouped state machine that can be controlled from a parent state machine. It does not work independently. There must be a state machine with state_machine_type of STATE_MACHINE_TYPE_ROOT or STATE_MACHINE_TYPE_NESTED in the parent or ancestor.

Property Descriptions

bool **allow_transition_to_self** = `false` 🔗

- void **set_allow_transition_to_self**(value: bool)
- bool **is_allow_transition_to_self**()

If `true`, allows teleport to the self state with AnimationNodeStateMachinePlayback.travel. When the reset option is enabled in AnimationNodeStateMachinePlayback.travel, the animation is restarted. If `false`, nothing happens on the teleportation to the self state.

---

bool **reset_ends** = `false` 🔗

- void **set_reset_ends**(value: bool)
- bool **are_ends_reset**()

If `true`, treat the cross-fade to the start and end nodes as a blend with the RESET animation.

In most cases, when additional cross-fades are performed in the parent AnimationNode of the state machine, setting this property to `false` and matching the cross-fade time of the parent AnimationNode and the state machine's start node and end node gives good results.

StateMachineType **state_machine_type** = `0`

- void **set_state_machine_type**(value: StateMachineType)
- StateMachineType **get_state_machine_type**()

This property can define the process of transitions for different use cases. See also StateMachineType.

Method Descriptions

void **add_node**(name: StringName, node: AnimationNode, position: Vector2 = Vector2(0, 0))

Adds a new animation node to the graph. The `position` is used for display in the editor.

---

void **add_transition**(from: StringName, to: StringName, transition: AnimationNodeStateMachineTransition)

Adds a transition between the given animation nodes.

Vector2 **get_graph_offset**() const

Returns the draw offset of the graph. Used for display in the editor.

AnimationNode **get_node**(name: StringName) const

Returns the animation node with the given name.

StringName **get_node_name**(node: AnimationNode) const

Returns the given animation node's name.

Vector2 **get_node_position**(name: StringName) const 🔗

Returns the given animation node's coordinates. Used for display in the editor.

AnimationNodeStateMachineTransition **get_transition**(idx: int) const 🔗

Returns the given transition.

---

int **get_transition_count**() const 🔗

Returns the number of connections in the graph.

StringName **get_transition_from**(idx: int) const 🔗

Returns the given transition's start node.

StringName **get_transition_to**(idx: int) const 🔗

Returns the given transition's end node.

---

bool **has_node**(name: StringName) const 🔗

Returns `true` if the graph contains the given animation node.

---

bool **has_transition**(from: StringName, to: StringName) const 🔗

Returns `true` if there is a transition between the given animation nodes.

---

void **remove_node**(name: StringName) 🔗

Deletes the given animation node from the graph.

---

void **remove_transition**(from: StringName, to: StringName) 🔗

Deletes the transition between the two specified animation nodes.

---

void **remove_transition_by_index**(idx: int)

Deletes the given transition by index.

---

void **rename_node**(name: StringName, new_name: StringName)

Renames the given animation node.

---

void **replace_node**(name: StringName, node: AnimationNode)

Replaces the given animation node with a new animation node.

---

void **set_graph_offset**(offset: Vector2)

Sets the draw offset of the graph. Used for display in the editor.

---

void **set_node_position**(name: StringName, position: Vector2)

Sets the animation node's coordinates. Used for display in the editor.

AnimationNodeStateMachinePlayback

Inherits: Resource **<** RefCounted **<** Object

Provides playback control for an AnimationNodeStateMachine.

Description

Allows control of AnimationTree state machines created with AnimationNodeStateMachine. Retrieve with `$AnimationTree.get("parameters/playback")`.

Example:

GDScript

```
var state_machine = $AnimationTree.get("parameters/playback")
state_machine.travel("some_state")
```

C#

```
var stateMachine = GetNode<AnimationTree>
("AnimationTree").Get("parameters/playback").As<AnimationNodeSt
ateMachinePlayback>();
stateMachine.Travel("some_state");
```

Tutorials

- Using AnimationTree

Properties

bool	resource_local_to_scene	`true`	(overrides Resource)

Methods

float	get_current_length() const
StringName	get_current_node() const
float	get_current_play_position() const
StringName	get_fading_from_node() const
Array[StringName]	get_travel_path() const
bool	is_playing() const
void	next()
void	start(node: StringName, reset: bool = true)
void	stop()

void	travel(to_node: StringName, reset_on_teleport: bool = true)

Method Descriptions

float **get_current_length**() const 🔗

Returns the current state length.

Note: It is possible that any AnimationRootNode can be nodes as well as animations. This means that there can be multiple animations within a single state. Which animation length has priority depends on the nodes connected inside it. Also, if a transition does not reset, the remaining length at that point will be returned.

StringName **get_current_node**() const 🔗

Returns the currently playing animation state.

Note: When using a cross-fade, the current state changes to the next state immediately after the cross-fade begins.

float **get_current_play_position**() const 🔗

Returns the playback position within the current animation state.

StringName **get_fading_from_node**() const 🔗

Returns the starting state of currently fading animation.

Array[StringName] **get_travel_path**() const

Returns the current travel path as computed internally by the A* algorithm.

---

bool **is_playing**() const

Returns `true` if an animation is playing.

---

void **next**()

If there is a next path by travel or auto advance, immediately transitions from the current state to the next state.

---

void **start**(node: StringName, reset: bool = true)

Starts playing the given animation.

If `reset` is `true`, the animation is played from the beginning.

---

void **stop**()

Stops the currently playing animation.

---

void **travel**(to_node: StringName, reset_on_teleport: bool = true)

Transitions from the current state to another one, following the shortest path.

If the path does not connect from the current state, the animation will play after the state teleports.

If `reset_on_teleport` is `true`, the animation is played from the beginning when the travel cause a teleportation.

AnimationNodeStateMachineTransition

Inherits: Resource **<** RefCounted **<** Object

A transition within an AnimationNodeStateMachine connecting two AnimationRootNodes.

Description

The path generated when using AnimationNodeStateMachinePlayback.travel is limited to the nodes connected by **AnimationNodeStateMachineTransition**.

You can set the timing and conditions of the transition in detail.

Tutorials

- Using AnimationTree

Properties

StringName	advance_condition	`&""`
String	advance_expression	`""`
AdvanceMode	advance_mode	`1`

bool	break_loop_at_end	`false`
int	priority	`1`
bool	reset	`true`
SwitchMode	switch_mode	`0`
Curve	xfade_curve	
float	xfade_time	`0.0`

Signals

advance_condition_changed() 🔗

Emitted when advance_condition is changed.

Enumerations

enum **SwitchMode**: 🔗

SwitchMode **SWITCH_MODE_IMMEDIATE** = `0`

Switch to the next state immediately. The current state will end and blend into the beginning of the new one.

SwitchMode **SWITCH_MODE_SYNC** = `1`

Switch to the next state immediately, but will seek the new state to the playback position of the old state.

SwitchMode **SWITCH_MODE_AT_END** = `2`

Wait for the current state playback to end, then switch to the beginning of the next state animation.

---

enum **AdvanceMode**: 🔗

AdvanceMode **ADVANCE_MODE_DISABLED** = `0`

Don't use this transition.

AdvanceMode **ADVANCE_MODE_ENABLED** = `1`

Only use this transition during AnimationNodeStateMachinePlayback.travel.

AdvanceMode **ADVANCE_MODE_AUTO** = `2`

Automatically use this transition if the advance_condition and advance_expression checks are true (if assigned).

Property Descriptions

StringName **advance_condition** = `&""` 🔗

- void **set_advance_condition**(value: StringName)
- StringName **get_advance_condition**()

Turn on auto advance when this condition is set. The provided name will become a boolean parameter on the AnimationTree that can be controlled from code (see Using AnimationTree). For example, if AnimationTree.tree_root is an AnimationNodeStateMachine and advance_condition is set to `"idle"`:

GDScript

```
$animation_tree.set("parameters/conditions/idle", is_on_floor
and (linear_velocity.x == 0))
```

C#

```
GetNode<AnimationTree>
("animation_tree").Set("parameters/conditions/idle", IsOnFloor
&& (LinearVelocity.X == 0));
```

String **advance_expression** = `""` 🔗

- void **set_advance_expression**(value: String)
- String **get_advance_expression**()

Use an expression as a condition for state machine transitions. It is possible to create complex animation advance conditions for switching between states and gives much greater flexibility for creating complex state machines by directly interfacing with the script code.

AdvanceMode **advance_mode** = `1` 🔗

- void **set_advance_mode**(value: AdvanceMode)
- AdvanceMode **get_advance_mode**()

Determines whether the transition should disabled, enabled when using AnimationNodeStateMachinePlayback.travel, or traversed automatically if the advance_condition and advance_expression checks are true (if assigned).

---

bool **break_loop_at_end** = `false`

- void **set_break_loop_at_end**(value: bool)
- bool **is_loop_broken_at_end**()

If `true`, breaks the loop at the end of the loop cycle for transition, even if the animation is looping.

---

int **priority** = `1`

- void **set_priority**(value: int)
- int **get_priority**()

Lower priority transitions are preferred when travelling through the tree via AnimationNodeStateMachinePlayback.travel or advance_mode is set to ADVANCE_MODE_AUTO.

---

bool **reset** = `true`

- void **set_reset**(value: bool)
- bool **is_reset**()

If `true`, the destination animation is played back from the beginning when switched.

SwitchMode **switch_mode** = `0`

- void **set_switch_mode**(value: SwitchMode)
- SwitchMode **get_switch_mode**()

The transition type.

Curve **xfade_curve** 🔗

- void **set_xfade_curve**(value: Curve)
- Curve **get_xfade_curve**()

Ease curve for better control over cross-fade between this state and the next.

---

float **xfade_time** = `0.0` 🔗

- void **set_xfade_time**(value: float)
- float **get_xfade_time**()

The time to cross-fade between this state and the next.

Note: AnimationNodeStateMachine transitions the current state immediately after the start of the fading. The precise remaining time can only be inferred from the main animation. When AnimationNodeOutput is considered as the most upstream, so the xfade_time is not scaled depending on the downstream delta. See also AnimationNodeOneShot.fadeout_time.

AnimationNodeSub2

Inherits: AnimationNodeSync **<** AnimationNode **<** Resource **<** RefCounted **<** Object

Blends two animations subtractively inside of an AnimationNodeBlendTree.

Description

A resource to add to an AnimationNodeBlendTree. Blends two animations subtractively based on the amount value.

This animation node is usually used for pre-calculation to cancel out any extra poses from the animation for the "add" animation source in AnimationNodeAdd2 or AnimationNodeAdd3.

In general, the blend value should be in the `[0.0, 1.0]` range, but values outside of this range can be used for amplified or inverted animations.

Note: This calculation is different from using a negative value in AnimationNodeAdd2, since the transformation matrices do not satisfy the commutative law. **AnimationNodeSub2** multiplies the transformation matrix of the inverted animation from the left side, while negative AnimationNodeAdd2 multiplies it from the right side.

Tutorials

- AnimationTree

AnimationNodeSync

Inherits: AnimationNode **<** Resource **<** RefCounted **<** Object

Inherited By: AnimationNodeAdd2, AnimationNodeAdd3, AnimationNodeBlend2, AnimationNodeBlend3, AnimationNodeOneShot, AnimationNodeSub2, AnimationNodeTransition

Base class for AnimationNodes with multiple input ports that must be synchronized.

Description

An animation node used to combine, mix, or blend two or more animations together while keeping them synchronized within an AnimationTree.

Tutorials

- Using AnimationTree

Properties

bool	sync	`false`

Property Descriptions

bool **sync** = `false`

- void **set_use_sync**(value: bool)
- bool **is_using_sync**()

If `false`, the blended animations' frame are stopped when the blend value is `0`.

If `true`, forcing the blended animations to advance frame.

AnimationNodeTimeScale

Inherits: AnimationNode **<** Resource **<** RefCounted **<** Object

A time-scaling animation node used in AnimationTree.

Description

Allows to scale the speed of the animation (or reverse it) in any child AnimationNodes. Setting it to `0.0` will pause the animation.

Tutorials

- Using AnimationTree
- 3D Platformer Demo [https://godotengine.org/asset-library/asset/2748]

AnimationNodeTimeSeek

Inherits: AnimationNode **<** Resource **<** RefCounted **<** Object

A time-seeking animation node used in AnimationTree.

Description

This animation node can be used to cause a seek command to happen to any sub-children of the animation graph. Use to play an Animation from the start or a certain playback position inside the AnimationNodeBlendTree.

After setting the time and changing the animation playback, the time seek node automatically goes into sleep mode on the next process frame by setting its `seek_request` value to `-1.0`.

GDScript

```
# Play child animation from the start.
animation_tree.set("parameters/TimeSeek/seek_request", 0.0)
# Alternative syntax (same result as above).
animation_tree["parameters/TimeSeek/seek_request"] = 0.0

# Play child animation from 12 second timestamp.
animation_tree.set("parameters/TimeSeek/seek_request", 12.0)
# Alternative syntax (same result as above).
animation_tree["parameters/TimeSeek/seek_request"] = 12.0
```

C#

```
// Play child animation from the start.
animationTree.Set("parameters/TimeSeek/seek_request", 0.0);
```

```
// Play child animation from 12 second timestamp.
animationTree.Set("parameters/TimeSeek/seek_request", 12.0);
```

Tutorials

- Using AnimationTree

AnimationNodeTransition

Inherits: AnimationNodeSync **<** AnimationNode **<** Resource **<** RefCounted **<** Object

A transition within an AnimationTree connecting two AnimationNodes.

Description

Simple state machine for cases which don't require a more advanced AnimationNodeStateMachine. Animations can be connected to the inputs and transition times can be specified.

After setting the request and changing the animation playback, the transition node automatically clears the request on the next process frame by setting its `transition_request` value to empty.

Note: When using a cross-fade, `current_state` and `current_index` change to the next state immediately after the cross-fade begins.

GDScript

```
# Play child animation connected to "state_2" port.
animation_tree.set("parameters/Transition/transition_request", "state_2")
# Alternative syntax (same result as above).
animation_tree["parameters/Transition/transition_request"] = "state_2"

# Get current state name (read-only).
animation_tree.get("parameters/Transition/current_state")
# Alternative syntax (same result as above).
animation_tree["parameters/Transition/current_state"]
```

```
# Get current state index (read-only).
animation_tree.get("parameters/Transition/current_index")
# Alternative syntax (same result as above).
animation_tree["parameters/Transition/current_index"]
```

C#

```
// Play child animation connected to "state_2" port.
animationTree.Set("parameters/Transition/transition_request",
"state_2");

// Get current state name (read-only).
animationTree.Get("parameters/Transition/current_state");

// Get current state index (read-only).
animationTree.Get("parameters/Transition/current_index");
```

Tutorials

- Using AnimationTree
- 3D Platformer Demo [https://godotengine.org/asset-library/asset/2748]
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

bool	allow_transition_to_self	`false`
int	input_count	`0`
Curve	xfade_curve	

float	xfade_time	`0.0`

Methods

bool	is_input_loop_broken_at_end(input: int) const
bool	is_input_reset(input: int) const
bool	is_input_set_as_auto_advance(input: int) const
void	set_input_as_auto_advance(input: int, enable: bool)
void	set_input_break_loop_at_end(input: int, enable: bool)
void	set_input_reset(input: int, enable: bool)

Property Descriptions

bool **allow_transition_to_self** = `false`

- void **set_allow_transition_to_self**(value: bool)
- bool **is_allow_transition_to_self**()

If `true`, allows transition to the self state. When the reset option is enabled in input, the animation is restarted. If `false`, nothing happens on the transition to the self state.

---

int **input_count** = `0`

- void **set_input_count**(value: int)
- int **get_input_count**()

The number of enabled input ports for this animation node.

Curve **xfade_curve**

- void **set_xfade_curve**(value: Curve)
- Curve **get_xfade_curve**()

Determines how cross-fading between animations is eased. If empty, the transition will be linear.

---

float **xfade_time** = `0.0`

- void **set_xfade_time**(value: float)
- float **get_xfade_time**()

Cross-fading time (in seconds) between each animation connected to the inputs.

Note: AnimationNodeTransition transitions the current state immediately after the start of the fading. The precise remaining time can only be inferred from the main animation. When AnimationNodeOutput is considered as the most upstream, so the xfade_time is not scaled depending on the downstream delta. See also AnimationNodeOneShot.fadeout_time.

Method Descriptions

bool **is_input_loop_broken_at_end**(input: int) const

Returns whether the animation breaks the loop at the end of the loop cycle for transition.

---

bool **is_input_reset**(input: int) const

Returns whether the animation restarts when the animation transitions from the other animation.

---

bool **is_input_set_as_auto_advance**(input: int) const

Returns `true` if auto-advance is enabled for the given `input` index.

---

void **set_input_as_auto_advance**(input: int, enable: bool)

Enables or disables auto-advance for the given `input` index. If enabled, state changes to the next input after playing the animation once. If enabled for the last input state, it loops to the first.

---

void **set_input_break_loop_at_end**(input: int, enable: bool)

If `true`, breaks the loop at the end of the loop cycle for transition, even if the animation is looping.

---

void **set_input_reset**(input: int, enable: bool)

If `true`, the destination animation is restarted when the animation transitions.

AnimationRootNode

Inherits: AnimationNode **<** Resource **<** RefCounted **<** Object

Inherited By: AnimationNodeAnimation, AnimationNodeBlendSpace1D, AnimationNodeBlendSpace2D, AnimationNodeBlendTree, AnimationNodeStateMachine

Base class for AnimationNodes that hold one or multiple composite animations. Usually used for AnimationTree.tree_root.

Description

AnimationRootNode is a base class for AnimationNodes that hold a complete animation. A complete animation refers to the output of an AnimationNodeOutput in an AnimationNodeBlendTree or the output of another **AnimationRootNode**. Used for AnimationTree.tree_root or in other **AnimationRootNode**s.

Examples of built-in root nodes include AnimationNodeBlendTree (allows blending nodes between each other using various modes), AnimationNodeStateMachine (allows to configure blending and transitions between nodes using a state machine pattern), AnimationNodeBlendSpace2D (allows linear blending between **three** AnimationNodes), AnimationNodeBlendSpace1D (allows linear blending only between **two** AnimationNodes).

Tutorials

- Using AnimationTree

ArrayMesh

Inherits: Mesh **<** Resource **<** RefCounted **<** Object

Mesh type that provides utility for constructing a surface from arrays.

Description

The **ArrayMesh** is used to construct a Mesh by specifying the attributes as arrays.

The most basic example is the creation of a single triangle:

GDScript

```
var vertices = PackedVector3Array()
vertices.push_back(Vector3(0, 1, 0))
vertices.push_back(Vector3(1, 0, 0))
vertices.push_back(Vector3(0, 0, 1))

# Initialize the ArrayMesh.
var arr_mesh = ArrayMesh.new()
var arrays = []
arrays.resize(Mesh.ARRAY_MAX)
arrays[Mesh.ARRAY_VERTEX] = vertices

# Create the Mesh.
arr_mesh.add_surface_from_arrays(Mesh.PRIMITIVE_TRIANGLES,
arrays)
var m = MeshInstance3D.new()
m.mesh = arr_mesh
```

C#

```
var vertices = new Vector3[]
{
    new Vector3(0, 1, 0),
    new Vector3(1, 0, 0),
    new Vector3(0, 0, 1),
};

// Initialize the ArrayMesh.
var arrMesh = new ArrayMesh();
var arrays = new Godot.Collections.Array();
arrays.Resize((int)Mesh.ArrayType.Max);
arrays[(int)Mesh.ArrayType.Vertex] = vertices;

// Create the Mesh.
arrMesh.AddSurfaceFromArrays(Mesh.PrimitiveType.Triangles,
arrays);
var m = new MeshInstance3D();
m.Mesh = arrMesh;
```

The MeshInstance3D is ready to be added to the SceneTree to be shown.

See also ImmediateMesh, MeshDataTool and SurfaceTool for procedural geometry generation.

Note: Godot uses clockwise winding order [https://learnopengl.com/Advanced-OpenGL/Face-culling] for front faces of triangle primitive modes.

Tutorials

- Procedural geometry using the ArrayMesh

Properties

BlendShapeMode	blend_shape_mode	1

AABB	custom_aabb	`AABB(0, 0, 0, 0, 0, 0)`
ArrayMesh	shadow_mesh	

Methods

void	add_blend_shape(name: StringName)
void	add_surface_from_arrays(primitive: PrimitiveType, arrays: Array, blend_shapes: Array[Array] = [], lods: Dictionary = {}, flags: BitField[ArrayFormat] = 0)
void	clear_blend_shapes()
void	clear_surfaces()
int	get_blend_shape_count() const
StringName	get_blend_shape_name(index: int) const
Error	lightmap_unwrap(transform: Transform3D, texel_size: float)
void	regen_normal_maps()

void	set_blend_shape_name(index: int, name: StringName)
int	surface_find_by_name(name: String) const
int	surface_get_array_index_len(surf_idx: int) const
int	surface_get_array_len(surf_idx: int) const
BitField[ArrayFormat]	surface_get_format(surf_idx: int) const
String	surface_get_name(surf_idx: int) const
PrimitiveType	surface_get_primitive_type(surf_idx: int) const
void	surface_set_name(surf_idx: int, name: String)
void	surface_update_attribute_region(surf_idx: int, offset: int, data: PackedByteArray)
void	surface_update_skin_region(surf_idx: int, offset: int, data: PackedByteArray)

void	surface_update_vertex_region(surf_idx: int, offset: int, data: PackedByteArray)

Property Descriptions

BlendShapeMode **blend_shape_mode** = `1`

- void **set_blend_shape_mode**(value: BlendShapeMode)
- BlendShapeMode **get_blend_shape_mode**()

Sets the blend shape mode to one of BlendShapeMode.

AABB **custom_aabb** = `AABB(0, 0, 0, 0, 0, 0)`

- void **set_custom_aabb**(value: AABB)
- AABB **get_custom_aabb**()

Overrides the AABB with one defined by user for use with frustum culling. Especially useful to avoid unexpected culling when using a shader to offset vertices.

ArrayMesh **shadow_mesh**

- void **set_shadow_mesh**(value: ArrayMesh)
- ArrayMesh **get_shadow_mesh**()

An optional mesh which can be used for rendering shadows and the depth prepass. Can be used to increase performance by supplying a mesh with fused vertices and only vertex position data (without normals, UVs, colors, etc.).

Note: This mesh must have exactly the same vertex positions as the source mesh (including the source mesh's LODs, if present). If vertex positions differ, then the mesh will not draw correctly.

Method Descriptions

void **add_blend_shape**(name: StringName) 🔗

Adds name for a blend shape that will be added with add_surface_from_arrays. Must be called before surface is added.

---

void **add_surface_from_arrays**(primitive: PrimitiveType, arrays: Array, blend_shapes: Array[Array] = [], lods: Dictionary = {}, flags: BitField[ArrayFormat] = 0) 🔗

Creates a new surface. Mesh.get_surface_count will become the `surf_idx` for this new surface.

Surfaces are created to be rendered using a `primitive`, which may be any of the values defined in PrimitiveType.

The `arrays` argument is an array of arrays. Each of the Mesh.ARRAY_MAX elements contains an array with some of the mesh data for this surface as described by the corresponding member of ArrayType or `null` if it is not used by the surface. For example, `arrays[0]` is the array of vertices. That first vertex sub-array is always required; the others are optional. Adding an index array puts this surface into "index mode" where the vertex and other arrays become the sources of data and the index array defines the vertex order. All sub-arrays must have the same length as the vertex array (or be an exact multiple of the vertex array's length, when multiple elements of a sub-array correspond to a single vertex) or be empty, except for Mesh.ARRAY_INDEX if it is used.

The `blend_shapes` argument is an array of vertex data for each blend shape. Each element is an array of the same structure as `arrays`, but Mesh.ARRAY_VERTEX, Mesh.ARRAY_NORMAL, and Mesh.ARRAY_TANGENT are set if and only if they are set in `arrays` and all other entries are `null`.

The `lods` argument is a dictionary with float keys and PackedInt32Array values. Each entry in the dictionary represents an LOD level of the surface, where the value is the Mesh.ARRAY_INDEX array to use for the LOD level and the key is roughly proportional to the distance at which the LOD stats being used. I.e., increasing the key of an LOD also increases the distance that the objects has to be from the camera before the LOD is used.

The `flags` argument is the bitwise or of, as required: One value of ArrayCustomFormat left shifted by `ARRAY_FORMAT_CUSTOMn_SHIFT` for each custom channel in use, Mesh.ARRAY_FLAG_USE_DYNAMIC_UPDATE, Mesh.ARRAY_FLAG_USE_8_BONE_WEIGHTS, or Mesh.ARRAY_FLAG_USES_EMPTY_VERTEX_ARRAY.

Note: When using indices, it is recommended to only use points, lines, or triangles.

void **clear_blend_shapes**() 🔗

Removes all blend shapes from this **ArrayMesh**.

void **clear_surfaces**() 🔗

Removes all surfaces from this **ArrayMesh**.

int **get_blend_shape_count**() const 🔗

Returns the number of blend shapes that the **ArrayMesh** holds.

StringName **get_blend_shape_name**(index: int) const

Returns the name of the blend shape at this index.

Error **lightmap_unwrap**(transform: Transform3D, texel_size: float)

Performs a UV unwrap on the **ArrayMesh** to prepare the mesh for lightmapping.

void **regen_normal_maps**()

Regenerates tangents for each of the **ArrayMesh**'s surfaces.

void **set_blend_shape_name**(index: int, name: StringName)

Sets the name of the blend shape at this index.

int **surface_find_by_name**(name: String) const

Returns the index of the first surface with this name held within this **ArrayMesh**. If none are found, -1 is returned.

int **surface_get_array_index_len**(surf_idx: int) const

Returns the length in indices of the index array in the requested surface (see add_surface_from_arrays).

int **surface_get_array_len**(surf_idx: int) const 🔗

Returns the length in vertices of the vertex array in the requested surface (see add_surface_from_arrays).

BitField[ArrayFormat] **surface_get_format**(surf_idx: int) const 🔗

Returns the format mask of the requested surface (see add_surface_from_arrays).

String **surface_get_name**(surf_idx: int) const 🔗

Gets the name assigned to this surface.

PrimitiveType **surface_get_primitive_type**(surf_idx: int) const 🔗

Returns the primitive type of the requested surface (see add_surface_from_arrays).

---

void **surface_set_name**(surf_idx: int, name: String) 🔗

Sets a name for a given surface.

---

void **surface_update_attribute_region**(surf_idx: int, offset: int, data: PackedByteArray) 🔗

There is currently no description for this method. Please help us by contributing one!

---

void **surface_update_skin_region**(surf_idx: int, offset: int, data: PackedByteArray) 🔗

There is currently no description for this method. Please help us by contributing one!

---

void **surface_update_vertex_region**(surf_idx: int, offset: int, data: PackedByteArray) 🔗

There is currently no description for this method. Please help us by contributing one!

ArrayOccluder3D

Inherits: Occluder3D **<** Resource **<** RefCounted **<** Object

3D polygon shape for use with occlusion culling in OccluderInstance3D.

Description

ArrayOccluder3D stores an arbitrary 3D polygon shape that can be used by the engine's occlusion culling system. This is analogous to ArrayMesh, but for occluders.

See OccluderInstance3D's documentation for instructions on setting up occlusion culling.

Tutorials

- Occlusion culling

Properties

PackedInt32Array	indices	`PackedInt32Array()`
PackedVector3Array	vertices	`PackedVector3Array()`

Methods

void	set_arrays(vertices: PackedVector3Array, indices: PackedInt32Array)

Property Descriptions

PackedInt32Array **indices** = `PackedInt32Array()`

- void **set_indices**(value: PackedInt32Array)
- PackedInt32Array **get_indices**()

The occluder's index position. Indices determine which points from the vertices array should be drawn, and in which order.

Note: The occluder is always updated after setting this value. If creating occluders procedurally, consider using set_arrays instead to avoid updating the occluder twice when it's created.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedInt32Array for more details.

PackedVector3Array **vertices** = `PackedVector3Array()`

- void **set_vertices**(value: PackedVector3Array)
- PackedVector3Array **get_vertices**()

The occluder's vertex positions in local 3D coordinates.

Note: The occluder is always updated after setting this value. If creating occluders procedurally, consider using set_arrays instead to avoid updating the occluder twice when it's created.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector3Array for more details.

Method Descriptions

void **set_arrays**(vertices: PackedVector3Array, indices: PackedInt32Array) 🔗

Sets indices and vertices, while updating the final occluder only once after both values are set.

AtlasTexture

Inherits: Texture2D **<** Texture **<** Resource **<** RefCounted **<** Object

A texture that crops out part of another Texture2D.

Description

Texture2D resource that draws only part of its atlas texture, as defined by the region. An additional margin can also be set, which is useful for small adjustments.

Multiple **AtlasTexture** resources can be cropped from the same atlas. Packing many smaller textures into a singular large texture helps to optimize video memory costs and render calls.

Note: **AtlasTexture** cannot be used in an AnimatedTexture, and may not tile properly in nodes such as TextureRect, when inside other **AtlasTexture** resources.

Properties

Texture2D	atlas	
bool	filter_clip	`false`
Rect2	margin	`Rect2(0, 0, 0, 0)`

Rect2	region	`Rect2(0, 0, 0, 0)`
bool	resource_local_to_scene	`false` (overrides Resource)

Property Descriptions

Texture2D **atlas** 🔗

- void **set_atlas**(value: Texture2D)
- Texture2D **get_atlas**()

The texture that contains the atlas. Can be any type inheriting from Texture2D, including another **AtlasTexture**.

---

bool **filter_clip** = `false` 🔗

- void **set_filter_clip**(value: bool)
- bool **has_filter_clip**()

If `true`, the area outside of the region is clipped to avoid bleeding of the surrounding texture pixels.

Rect2 **margin** = `Rect2(0, 0, 0, 0)` 🔗

- void **set_margin**(value: Rect2)
- Rect2 **get_margin**()

The margin around the region. Useful for small adjustments. If the Rect2.size of this property ("w" and "h" in the editor) is set, the drawn texture is resized to fit within the margin.

Rect2 **region** = `Rect2(0, 0, 0, 0)` 🔗

- void **set_region**(value: Rect2)
- Rect2 **get_region**()

The region used to draw the atlas. If either dimension of the region's size is `0`, the value from atlas size will be used for that axis instead.

AudioBusLayout

Inherits: Resource **<** RefCounted **<** Object

Stores information about the audio buses.

Description

Stores position, muting, solo, bypass, effects, effect position, volume, and the connections between buses. See AudioServer for usage.

AudioEffect

Inherits: Resource **<** RefCounted **<** Object

Inherited By: AudioEffectAmplify, AudioEffectCapture, AudioEffectChorus, AudioEffectCompressor, AudioEffectDelay, AudioEffectDistortion, AudioEffectEQ, AudioEffectFilter, AudioEffectHardLimiter, AudioEffectLimiter, AudioEffectPanner, AudioEffectPhaser, AudioEffectPitchShift, AudioEffectRecord, AudioEffectReverb, AudioEffectSpectrumAnalyzer, AudioEffectStereoEnhance

Base class for audio effect resources.

Description

The base Resource for every audio effect. In the editor, an audio effect can be added to the current bus layout through the Audio panel. At run-time, it is also possible to manipulate audio effects through AudioServer.add_bus_effect, AudioServer.remove_bus_effect, and AudioServer.get_bus_effect.

When applied on a bus, an audio effect creates a corresponding AudioEffectInstance. The instance is directly responsible for manipulating the sound, based on the original audio effect's properties.

Tutorials

- Audio buses
- Audio Microphone Record Demo [https://godotengine.org/asset-library/asset/2760]

Methods

AudioEffectInstance	_instantiate() virtual

Method Descriptions

AudioEffectInstance **_instantiate**() virtual

Override this method to customize the AudioEffectInstance created when this effect is applied on a bus in the editor's Audio panel, or through AudioServer.add_bus_effect.

```
extends AudioEffect

@export var strength = 4.0

func _instantiate():
    var effect = CustomAudioEffectInstance.new()
    effect.base = self

    return effect
```

Note: It is recommended to keep a reference to the original **AudioEffect** in the new instance. Depending on the implementation this allows the effect instance to listen for changes at run-time and be modified accordingly.

AudioEffectAmplify

Inherits: AudioEffect **<** Resource **<** RefCounted **<** Object

Adds an amplifying audio effect to an audio bus.

Description

Increases or decreases the volume being routed through the audio bus.

Tutorials

- Audio buses

Properties

float	volume_db	`0.0`

Property Descriptions

float **volume_db** = `0.0` 🔗

- void **set_volume_db**(value: float)
- float **get_volume_db**()

Amount of amplification in decibels. Positive values make the sound louder, negative values make it quieter. Value can range from -80 to 24.

AudioEffectBandLimitFilter

Inherits: AudioEffectFilter **<** AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a band limit filter to the audio bus.

Description

Limits the frequencies in a range around the AudioEffectFilter.cutoff_hz and allows frequencies outside of this range to pass.

Tutorials

- Audio buses

AudioEffectBandPassFilter

Inherits: AudioEffectFilter **<** AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a band pass filter to the audio bus.

Description

Attenuates the frequencies inside of a range around the AudioEffectFilter.cutoff_hz and cuts frequencies outside of this band.

Tutorials

- Audio buses

AudioEffectCapture

Inherits: AudioEffect **<** Resource **<** RefCounted **<** Object

Captures audio from an audio bus in real-time.

Description

AudioEffectCapture is an AudioEffect which copies all audio frames from the attached audio effect bus into its internal ring buffer.

Application code should consume these audio frames from this ring buffer using get_buffer and process it as needed, for example to capture data from an AudioStreamMicrophone, implement application-defined effects, or to transmit audio over the network. When capturing audio data from a microphone, the format of the samples will be stereo 32-bit floating-point PCM.

Unlike AudioEffectRecord, this effect only returns the raw audio samples instead of encoding them into an AudioStream.

Tutorials

- Audio buses

Properties

float	buffer_length	`0.1`

Methods

bool	can_get_buffer(frames: int) const
void	clear_buffer()
PackedVector2Array	get_buffer(frames: int)
int	get_buffer_length_frames() const
int	get_discarded_frames() const
int	get_frames_available() const
int	get_pushed_frames() const

Property Descriptions

float **buffer_length** = `0.1`

- void **set_buffer_length**(value: float)
- float **get_buffer_length**()

Length of the internal ring buffer, in seconds. Setting the buffer length will have no effect if already initialized.

Method Descriptions

bool **can_get_buffer**(frames: int) const

Returns `true` if at least `frames` audio frames are available to read in the internal ring buffer.

---

void **clear_buffer**()

Clears the internal ring buffer.

Note: Calling this during a capture can cause the loss of samples which causes popping in the playback.

PackedVector2Array **get_buffer**(frames: int)

Gets the next `frames` audio samples from the internal ring buffer.

Returns a PackedVector2Array containing exactly `frames` audio samples if available, or an empty PackedVector2Array if insufficient data was available.

The samples are signed floating-point PCM between `-1` and `1`. You will have to scale them if you want to use them as 8 or 16-bit integer samples. (`v = 0x7fff * samples[0].x`)

---

int **get_buffer_length_frames**() const

Returns the total size of the internal ring buffer in frames.

---

int **get_discarded_frames**() const

Returns the number of audio frames discarded from the audio bus due to full buffer.

---

int **get_frames_available**() const

Returns the number of frames available to read using get_buffer.

---

int **get_pushed_frames**() const

Returns the number of audio frames inserted from the audio bus.

AudioEffectChorus

Inherits: AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a chorus audio effect.

Description

Adds a chorus audio effect. The effect applies a filter with voices to duplicate the audio source and manipulate it through the filter.

Tutorials

- Audio buses

Properties

float	dry	`1.0`
float	voice/1/cutoff_hz	`8000.0`
float	voice/1/delay_ms	`15.0`
float	voice/1/depth_ms	`2.0`
float	voice/1/level_db	`0.0`

float	voice/1/pan	`-0.5`
float	voice/1/rate_hz	`0.8`
float	voice/2/cutoff_hz	`8000.0`
float	voice/2/delay_ms	`20.0`
float	voice/2/depth_ms	`3.0`
float	voice/2/level_db	`0.0`
float	voice/2/pan	`0.5`
float	voice/2/rate_hz	`1.2`
float	voice/3/cutoff_hz	
float	voice/3/delay_ms	
float	voice/3/depth_ms	
float	voice/3/level_db	
float	voice/3/pan	

float	voice/3/rate_hz	
float	voice/4/cutoff_hz	
float	voice/4/delay_ms	
float	voice/4/depth_ms	
float	voice/4/level_db	
float	voice/4/pan	
float	voice/4/rate_hz	
int	voice_count	`2`
float	wet	`0.5`

Methods

float	get_voice_cutoff_hz(voice_idx: int) const
float	get_voice_delay_ms(voice_idx: int) const

float	get_voice_depth_ms(voice_idx: int) const
float	get_voice_level_db(voice_idx: int) const
float	get_voice_pan(voice_idx: int) const
float	get_voice_rate_hz(voice_idx: int) const
void	set_voice_cutoff_hz(voice_idx: int, cutoff_hz: float)
void	set_voice_delay_ms(voice_idx: int, delay_ms: float)
void	set_voice_depth_ms(voice_idx: int, depth_ms: float)
void	set_voice_level_db(voice_idx: int, level_db: float)
void	set_voice_pan(voice_idx: int, pan: float)
void	set_voice_rate_hz(voice_idx: int, rate_hz: float)

Property Descriptions

float **dry** = `1.0`

- void **set_dry**(value: float)

- float **get_dry**()

The effect's raw signal.

---

float **voice/1/cutoff_hz** = `8000.0`

- void **set_voice_cutoff_hz**(voice_idx: int, cutoff_hz: float)
- float **get_voice_cutoff_hz**(voice_idx: int) const

The voice's cutoff frequency.

---

float **voice/1/delay_ms** = `15.0`

- void **set_voice_delay_ms**(voice_idx: int, delay_ms: float)
- float **get_voice_delay_ms**(voice_idx: int) const

The voice's signal delay.

---

float **voice/1/depth_ms** = `2.0`

- void **set_voice_depth_ms**(voice_idx: int, depth_ms: float)
- float **get_voice_depth_ms**(voice_idx: int) const

The voice filter's depth.

---

float **voice/1/level_db** = `0.0`

- void **set_voice_level_db**(voice_idx: int, level_db: float)
- float **get_voice_level_db**(voice_idx: int) const

The voice's volume.

---

float **voice/1/pan** = `-0.5`

- void **set_voice_pan**(voice_idx: int, pan: float)
- float **get_voice_pan**(voice_idx: int) const

The voice's pan level.

---

float **voice/1/rate_hz** = `0.8`

- void **set_voice_rate_hz**(voice_idx: int, rate_hz: float)
- float **get_voice_rate_hz**(voice_idx: int) const

The voice's filter rate.

---

float **voice/2/cutoff_hz** = `8000.0`

- void **set_voice_cutoff_hz**(voice_idx: int, cutoff_hz: float)
- float **get_voice_cutoff_hz**(voice_idx: int) const

The voice's cutoff frequency.

---

float **voice/2/delay_ms** = `20.0`

- void **set_voice_delay_ms**(voice_idx: int, delay_ms: float)
- float **get_voice_delay_ms**(voice_idx: int) const

The voice's signal delay.

---

float **voice/2/depth_ms** = `3.0`

- void **set_voice_depth_ms**(voice_idx: int, depth_ms: float)
- float **get_voice_depth_ms**(voice_idx: int) const

The voice filter's depth.

---

float **voice/2/level_db** = `0.0`

- void **set_voice_level_db**(voice_idx: int, level_db: float)
- float **get_voice_level_db**(voice_idx: int) const

The voice's volume.

---

float **voice/2/pan** = `0.5`

- void **set_voice_pan**(voice_idx: int, pan: float)
- float **get_voice_pan**(voice_idx: int) const

The voice's pan level.

---

float **voice/2/rate_hz** = `1.2`

- void **set_voice_rate_hz**(voice_idx: int, rate_hz: float)
- float **get_voice_rate_hz**(voice_idx: int) const

The voice's filter rate.

---

float **voice/3/cutoff_hz**

- void **set_voice_cutoff_hz**(voice_idx: int, cutoff_hz: float)
- float **get_voice_cutoff_hz**(voice_idx: int) const

The voice's cutoff frequency.

---

float **voice/3/delay_ms**

- void **set_voice_delay_ms**(voice_idx: int, delay_ms: float)
- float **get_voice_delay_ms**(voice_idx: int) const

The voice's signal delay.

---

float **voice/3/depth_ms** 🔗

- void **set_voice_depth_ms**(voice_idx: int, depth_ms: float)
- float **get_voice_depth_ms**(voice_idx: int) const

The voice filter's depth.

---

float **voice/3/level_db** 🔗

- void **set_voice_level_db**(voice_idx: int, level_db: float)
- float **get_voice_level_db**(voice_idx: int) const

The voice's volume.

---

float **voice/3/pan** 🔗

- void **set_voice_pan**(voice_idx: int, pan: float)
- float **get_voice_pan**(voice_idx: int) const

The voice's pan level.

---

float **voice/3/rate_hz** 🔗

- void **set_voice_rate_hz**(voice_idx: int, rate_hz: float)
- float **get_voice_rate_hz**(voice_idx: int) const

The voice's filter rate.

float **voice/4/cutoff_hz**

- void **set_voice_cutoff_hz**(voice_idx: int, cutoff_hz: float)
- float **get_voice_cutoff_hz**(voice_idx: int) const

The voice's cutoff frequency.

---

float **voice/4/delay_ms**

- void **set_voice_delay_ms**(voice_idx: int, delay_ms: float)
- float **get_voice_delay_ms**(voice_idx: int) const

The voice's signal delay.

---

float **voice/4/depth_ms**

- void **set_voice_depth_ms**(voice_idx: int, depth_ms: float)
- float **get_voice_depth_ms**(voice_idx: int) const

The voice filter's depth.

---

float **voice/4/level_db**

- void **set_voice_level_db**(voice_idx: int, level_db: float)
- float **get_voice_level_db**(voice_idx: int) const

The voice's volume.

---

float **voice/4/pan**

- void **set_voice_pan**(voice_idx: int, pan: float)
- float **get_voice_pan**(voice_idx: int) const

The voice's pan level.

---

float **voice/4/rate_hz**

- void **set_voice_rate_hz**(voice_idx: int, rate_hz: float)
- float **get_voice_rate_hz**(voice_idx: int) const

The voice's filter rate.

---

int **voice_count** = `2`

- void **set_voice_count**(value: int)
- int **get_voice_count**()

The number of voices in the effect.

---

float **wet** = `0.5`

- void **set_wet**(value: float)
- float **get_wet**()

The effect's processed signal.

Method Descriptions

float **get_voice_cutoff_hz**(voice_idx: int) const

There is currently no description for this method. Please help us by contributing one!

---

float **get_voice_delay_ms**(voice_idx: int) const

There is currently no description for this method. Please help us by contributing one!

---

float **get_voice_depth_ms**(voice_idx: int) const

There is currently no description for this method. Please help us by contributing one!

---

float **get_voice_level_db**(voice_idx: int) const

There is currently no description for this method. Please help us by contributing one!

---

float **get_voice_pan**(voice_idx: int) const

There is currently no description for this method. Please help us by contributing one!

---

float **get_voice_rate_hz**(voice_idx: int) const

There is currently no description for this method. Please help us by contributing one!

---

void **set_voice_cutoff_hz**(voice_idx: int, cutoff_hz: float)

There is currently no description for this method. Please help us by contributing one!

---

void **set_voice_delay_ms**(voice_idx: int, delay_ms: float)

There is currently no description for this method. Please help us by contributing one!

---

void **set_voice_depth_ms**(voice_idx: int, depth_ms: float) 🔗

There is currently no description for this method. Please help us by contributing one!

---

void **set_voice_level_db**(voice_idx: int, level_db: float) 🔗

There is currently no description for this method. Please help us by contributing one!

---

void **set_voice_pan**(voice_idx: int, pan: float) 🔗

There is currently no description for this method. Please help us by contributing one!

---

void **set_voice_rate_hz**(voice_idx: int, rate_hz: float) 🔗

There is currently no description for this method. Please help us by contributing one!

AudioEffectCompressor

Inherits: AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a compressor audio effect to an audio bus.

Reduces sounds that exceed a certain threshold level, smooths out the dynamics and increases the overall volume.

Description

Dynamic range compressor reduces the level of the sound when the amplitude goes over a certain threshold in Decibels. One of the main uses of a compressor is to increase the dynamic range by clipping as little as possible (when sound goes over 0dB).

Compressor has many uses in the mix:

- In the Master bus to compress the whole output (although an AudioEffectLimiter is probably better).
- In voice channels to ensure they sound as balanced as possible.
- Sidechained. This can reduce the sound level sidechained with another audio bus for threshold detection. This technique is common in video game mixing to the level of music and SFX while voices are being heard.
- Accentuates transients by using a wider attack, making effects sound more punchy.

Tutorials

- Audio buses

Properties

float	attack_us	`20.0`
float	gain	`0.0`
float	mix	`1.0`
float	ratio	`4.0`
float	release_ms	`250.0`
StringName	sidechain	`&""`
float	threshold	`0.0`

Property Descriptions

float **attack_us** = `20.0` 🔗

- void **set_attack_us**(value: float)
- float **get_attack_us**()

Compressor's reaction time when the signal exceeds the threshold, in microseconds. Value can range from 20 to 2000.

---

float **gain** = `0.0` 🔗

- void **set_gain**(value: float)
- float **get_gain**()

Gain applied to the output signal.

---

float **mix** = `1.0` 🔗

- void **set_mix**(value: float)
- float **get_mix**()

Balance between original signal and effect signal. Value can range from 0 (totally dry) to 1 (totally wet).

---

float **ratio** = `4.0` 🔗

- void **set_ratio**(value: float)
- float **get_ratio**()

Amount of compression applied to the audio once it passes the threshold level. The higher the ratio, the more the loud parts of the audio will be compressed. Value can range from 1 to 48.

---

float **release_ms** = `250.0` 🔗

- void **set_release_ms**(value: float)
- float **get_release_ms**()

Compressor's delay time to stop reducing the signal after the signal level falls below the threshold, in milliseconds. Value can range from 20 to 2000.

StringName **sidechain** = `&""` 🔗

- void **set_sidechain**(value: StringName)
- StringName **get_sidechain**()

Reduce the sound level using another audio bus for threshold detection.

---

float **threshold** = `0.0` 🔗

- void **set_threshold**(value: float)
- float **get_threshold**()

The level above which compression is applied to the audio. Value can range from -60 to 0.

AudioEffectDelay

Inherits: AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a delay audio effect to an audio bus. Plays input signal back after a period of time.

Two tap delay and feedback options.

Description

Plays input signal back after a period of time. The delayed signal may be played back multiple times to create the sound of a repeating, decaying echo. Delay effects range from a subtle echo effect to a pronounced blending of previous sounds with new sounds.

Tutorials

- Audio buses

Properties

float	dry	`1.0`
bool	feedback_active	`false`
float	feedback_delay_ms	`340.0`

float	feedback_level_db	`-6.0`
float	feedback_lowpass	`16000.0`
bool	tap1_active	`true`
float	tap1_delay_ms	`250.0`
float	tap1_level_db	`-6.0`
float	tap1_pan	`0.2`
bool	tap2_active	`true`
float	tap2_delay_ms	`500.0`
float	tap2_level_db	`-12.0`
float	tap2_pan	`-0.4`

Property Descriptions

float **dry** = `1.0` 🔗

- void **set_dry**(value: float)
- float **get_dry**()

Output percent of original sound. At 0, only delayed sounds are output. Value can range from 0 to 1.

---

float **feedback_active** = `false`

- void **set_feedback_active**(value: bool)
- bool **is_feedback_active**()

If `true`, feedback is enabled.

---

float **feedback_delay_ms** = `340.0`

- void **set_feedback_delay_ms**(value: float)
- float **get_feedback_delay_ms**()

Feedback delay time in milliseconds.

---

float **feedback_level_db** = `-6.0`

- void **set_feedback_level_db**(value: float)
- float **get_feedback_level_db**()

Sound level for feedback.

---

float **feedback_lowpass** = `16000.0`

- void **set_feedback_lowpass**(value: float)
- float **get_feedback_lowpass**()

Low-pass filter for feedback, in Hz. Frequencies below this value are filtered out of the source signal.

---

bool **tap1_active** = `true` 🔗

- void **set_tap1_active**(value: bool)
- bool **is_tap1_active**()

If `true`, the first tap will be enabled.

---

float **tap1_delay_ms** = `250.0` 🔗

- void **set_tap1_delay_ms**(value: float)
- float **get_tap1_delay_ms**()

First tap delay time in milliseconds.

---

float **tap1_level_db** = `-6.0` 🔗

- void **set_tap1_level_db**(value: float)
- float **get_tap1_level_db**()

Sound level for the first tap.

---

float **tap1_pan** = `0.2` 🔗

- void **set_tap1_pan**(value: float)
- float **get_tap1_pan**()

Pan position for the first tap. Value can range from -1 (fully left) to 1 (fully right).

---

bool **tap2_active** = `true`

- void **set_tap2_active**(value: bool)
- bool **is_tap2_active**()

If `true`, the second tap will be enabled.

---

float **tap2_delay_ms** = `500.0`

- void **set_tap2_delay_ms**(value: float)
- float **get_tap2_delay_ms**()

Second tap delay time in milliseconds.

---

float **tap2_level_db** = `-12.0`

- void **set_tap2_level_db**(value: float)
- float **get_tap2_level_db**()

Sound level for the second tap.

---

float **tap2_pan** = `-0.4`

- void **set_tap2_pan**(value: float)
- float **get_tap2_pan**()

Pan position for the second tap. Value can range from -1 (fully left) to 1 (fully right).

AudioEffectDistortion

Inherits: AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a distortion audio effect to an Audio bus.

Modifies the sound to make it distorted.

Description

Different types are available: clip, tan, lo-fi (bit crushing), overdrive, or waveshape.

By distorting the waveform the frequency content changes, which will often make the sound "crunchy" or "abrasive". For games, it can simulate sound coming from some saturated device or speaker very efficiently.

Tutorials

- Audio buses

Properties

float	drive	`0.0`
float	keep_hf_hz	`16000.0`

Mode	mode	`0`
float	post_gain	`0.0`
float	pre_gain	`0.0`

Enumerations

enum **Mode**:

Mode **MODE_CLIP** = `0`

Digital distortion effect which cuts off peaks at the top and bottom of the waveform.

Mode **MODE_ATAN** = `1`

There is currently no description for this enum. Please help us by contributing one!

Mode **MODE_LOFI** = `2`

Low-resolution digital distortion effect (bit depth reduction). You can use it to emulate the sound of early digital audio devices.

Mode **MODE_OVERDRIVE** = `3`

Emulates the warm distortion produced by a field effect transistor, which is commonly used in solid-state musical instrument amplifiers. The drive property has no effect in this mode.

Mode **MODE_WAVESHAPE** = `4`

Waveshaper distortions are used mainly by electronic musicians to achieve an extra-abrasive sound.

Property Descriptions

float **drive** = `0.0`

- void **set_drive**(value: float)
- float **get_drive**()

Distortion power. Value can range from 0 to 1.

---

float **keep_hf_hz** = `16000.0`

- void **set_keep_hf_hz**(value: float)
- float **get_keep_hf_hz**()

High-pass filter, in Hz. Frequencies higher than this value will not be affected by the distortion. Value can range from 1 to 20000.

Mode **mode** = `0`

- void **set_mode**(value: Mode)
- Mode **get_mode**()

Distortion type.

---

float **post_gain** = `0.0`

- void **set_post_gain**(value: float)
- float **get_post_gain**()

Increases or decreases the volume after the effect, in decibels. Value can range from -80 to 24.

---

float **pre_gain** = `0.0` 🔗

- void **set_pre_gain**(value: float)
- float **get_pre_gain**()

Increases or decreases the volume before the effect, in decibels. Value can range from -60 to 60.

AudioEffectEQ

Inherits: AudioEffect **<** Resource **<** RefCounted **<** Object

Inherited By: AudioEffectEQ10, AudioEffectEQ21, AudioEffectEQ6

Base class for audio equalizers. Gives you control over frequencies.

Use it to create a custom equalizer if AudioEffectEQ6, AudioEffectEQ10 or AudioEffectEQ21 don't fit your needs.

Description

AudioEffectEQ gives you control over frequencies. Use it to compensate for existing deficiencies in audio. AudioEffectEQs are useful on the Master bus to completely master a mix and give it more character. They are also useful when a game is run on a mobile device, to adjust the mix to that kind of speakers (it can be added but disabled when headphones are plugged).

Tutorials

- Audio buses

Methods

int	get_band_count() const
float	get_band_gain_db(band_idx: int) const

void	set_band_gain_db(band_idx: int, volume_db: float)

Method Descriptions

int **get_band_count**() const

Returns the number of bands of the equalizer.

float **get_band_gain_db**(band_idx: int) const

Returns the band's gain at the specified index, in dB.

void **set_band_gain_db**(band_idx: int, volume_db: float)

Sets band's gain at the specified index, in dB.

AudioEffectEQ10

Inherits: AudioEffectEQ **<** AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a 10-band equalizer audio effect to an Audio bus. Gives you control over frequencies from 31 Hz to 16000 Hz.

Each frequency can be modulated between -60/+24 dB.

Description

Frequency bands:

Band 1: 31 Hz

Band 2: 62 Hz

Band 3: 125 Hz

Band 4: 250 Hz

Band 5: 500 Hz

Band 6: 1000 Hz

Band 7: 2000 Hz

Band 8: 4000 Hz

Band 9: 8000 Hz

Band 10: 16000 Hz

See also AudioEffectEQ, AudioEffectEQ6, AudioEffectEQ21.

Tutorials

- Audio buses

AudioEffectEQ21

Inherits: AudioEffectEQ **<** AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a 21-band equalizer audio effect to an Audio bus. Gives you control over frequencies from 22 Hz to 22000 Hz.

Each frequency can be modulated between -60/+24 dB.

Description

Frequency bands:

Band 1: 22 Hz

Band 2: 32 Hz

Band 3: 44 Hz

Band 4: 63 Hz

Band 5: 90 Hz

Band 6: 125 Hz

Band 7: 175 Hz

Band 8: 250 Hz

Band 9: 350 Hz

Band 10: 500 Hz

Band 11: 700 Hz

Band 12: 1000 Hz

Band 13: 1400 Hz

Band 14: 2000 Hz

Band 15: 2800 Hz

Band 16: 4000 Hz

Band 17: 5600 Hz

Band 18: 8000 Hz

Band 19: 11000 Hz

Band 20: 16000 Hz

Band 21: 22000 Hz

See also AudioEffectEQ, AudioEffectEQ6, AudioEffectEQ10.

Tutorials

- Audio buses

AudioEffectEQ6

Inherits: AudioEffectEQ **<** AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a 6-band equalizer audio effect to an audio bus. Gives you control over frequencies from 32 Hz to 10000 Hz.

Each frequency can be modulated between -60/+24 dB.

Description

Frequency bands:

Band 1: 32 Hz

Band 2: 100 Hz

Band 3: 320 Hz

Band 4: 1000 Hz

Band 5: 3200 Hz

Band 6: 10000 Hz

See also AudioEffectEQ, AudioEffectEQ10, AudioEffectEQ21.

Tutorials

- Audio buses

AudioEffectFilter

Inherits: AudioEffect **<** Resource **<** RefCounted **<** Object

Inherited By: AudioEffectBandLimitFilter, AudioEffectBandPassFilter, AudioEffectHighPassFilter, AudioEffectHighShelfFilter, AudioEffectLowPassFilter, AudioEffectLowShelfFilter, AudioEffectNotchFilter

Adds a filter to the audio bus.

Description

Allows frequencies other than the cutoff_hz to pass.

Tutorials

- Audio buses

Properties

float	cutoff_hz	`2000.0`
FilterDB	db	`0`
float	gain	`1.0`

float	resonance	`0.5`

Enumerations

enum **FilterDB**:

FilterDB **FILTER_6DB** = `0`

There is currently no description for this enum. Please help us by contributing one!

FilterDB **FILTER_12DB** = `1`

There is currently no description for this enum. Please help us by contributing one!

FilterDB **FILTER_18DB** = `2`

There is currently no description for this enum. Please help us by contributing one!

FilterDB **FILTER_24DB** = `3`

There is currently no description for this enum. Please help us by contributing one!

Property Descriptions

float **cutoff_hz** = `2000.0`

- void **set_cutoff**(value: float)

- float **get_cutoff**()

Threshold frequency for the filter, in Hz.

FilterDB **db** = `0`

- void **set_db**(value: FilterDB)
- FilterDB **get_db**()

There is currently no description for this property. Please help us by contributing one!

float **gain** = `1.0`

- void **set_gain**(value: float)
- float **get_gain**()

Gain amount of the frequencies after the filter.

float **resonance** = `0.5`

- void **set_resonance**(value: float)
- float **get_resonance**()

Amount of boost in the frequency range near the cutoff frequency.

AudioEffectHardLimiter

Inherits: AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a hard limiter audio effect to an Audio bus.

Description

A limiter is an effect designed to disallow sound from going over a given dB threshold. Hard limiters predict volume peaks, and will smoothly apply gain reduction when a peak crosses the ceiling threshold to prevent clipping and distortion. It preserves the waveform and prevents it from crossing the ceiling threshold. Adding one in the Master bus is recommended as a safety measure to prevent sudden volume peaks from occurring, and to prevent distortion caused by clipping.

Tutorials

- Audio buses

Properties

float	ceiling_db	`-0.3`
float	pre_gain_db	`0.0`
float	release	`0.1`

Property Descriptions

float **ceiling_db** = `-0.3` 🔗

- void **set_ceiling_db**(value: float)
- float **get_ceiling_db**()

The waveform's maximum allowed value, in decibels. This value can range from `-24.0` to `0.0`.

The default value of `-0.3` prevents potential inter-sample peaks (ISP) from crossing over 0 dB, which can cause slight distortion on some older hardware.

---

float **pre_gain_db** = `0.0` 🔗

- void **set_pre_gain_db**(value: float)
- float **get_pre_gain_db**()

Gain to apply before limiting, in decibels.

---

float **release** = `0.1` 🔗

- void **set_release**(value: float)
- float **get_release**()

Time it takes in seconds for the gain reduction to fully release.

AudioEffectHighPassFilter

Inherits: AudioEffectFilter **<** AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a high-pass filter to the audio bus.

Description

Cuts frequencies lower than the AudioEffectFilter.cutoff_hz and allows higher frequencies to pass.

Tutorials

- Audio buses

AudioEffectHighShelfFilter

Inherits: AudioEffectFilter **<** AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a high-shelf filter to the audio bus.

Description

Reduces all frequencies above the AudioEffectFilter.cutoff_hz.

Tutorials

- Audio buses

AudioEffectLimiter

Deprecated: Use AudioEffectHardLimiter instead.

Inherits: AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a soft-clip limiter audio effect to an Audio bus.

Description

A limiter is similar to a compressor, but it's less flexible and designed to disallow sound going over a given dB threshold. Adding one in the Master bus is always recommended to reduce the effects of clipping.

Soft clipping starts to reduce the peaks a little below the threshold level and progressively increases its effect as the input level increases such that the threshold is never exceeded.

Tutorials

- Audio buses

Properties

float	ceiling_db	`-0.1`
float	soft_clip_db	`2.0`

float	soft_clip_ratio	`10.0`
float	threshold_db	`0.0`

Property Descriptions

float **ceiling_db** = `-0.1`

- void **set_ceiling_db**(value: float)
- float **get_ceiling_db**()

The waveform's maximum allowed value, in decibels. Value can range from -20 to -0.1.

---

float **soft_clip_db** = `2.0`

- void **set_soft_clip_db**(value: float)
- float **get_soft_clip_db**()

Applies a gain to the limited waves, in decibels. Value can range from 0 to 6.

---

float **soft_clip_ratio** = `10.0`

- void **set_soft_clip_ratio**(value: float)
- float **get_soft_clip_ratio**()

There is currently no description for this property. Please help us by contributing one!

---

float **threshold_db** = `0.0` 🔗

- void **set_threshold_db**(value: float)
- float **get_threshold_db**()

Threshold from which the limiter begins to be active, in decibels. Value can range from -30 to 0.

AudioEffectLowPassFilter

Inherits: AudioEffectFilter **<** AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a low-pass filter to the audio bus.

Description

Cuts frequencies higher than the AudioEffectFilter.cutoff_hz and allows lower frequencies to pass.

Tutorials

- Audio buses

AudioEffectLowShelfFilter

Inherits: AudioEffectFilter **<** AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a low-shelf filter to the audio bus.

Description

Reduces all frequencies below the AudioEffectFilter.cutoff_hz.

Tutorials

- Audio buses

AudioEffectNotchFilter

Inherits: AudioEffectFilter **<** AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a notch filter to the Audio bus.

Description

Attenuates frequencies in a narrow band around the AudioEffectFilter.cutoff_hz and cuts frequencies outside of this range.

Tutorials

- Audio buses

AudioEffectPanner

Inherits: AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a panner audio effect to an audio bus. Pans sound left or right.

Description

Determines how much of an audio signal is sent to the left and right buses.

Tutorials

- Audio buses

Properties

float	pan	`0.0`

Property Descriptions

float **pan** = `0.0` 🔗

- void **set_pan**(value: float)
- float **get_pan**()

Pan position. Value can range from -1 (fully left) to 1 (fully right).

AudioEffectPhaser

Inherits: AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a phaser audio effect to an audio bus.

Combines the original signal with a copy that is slightly out of phase with the original.

Description

Combines phase-shifted signals with the original signal. The movement of the phase-shifted signals is controlled using a low-frequency oscillator.

Tutorials

- Audio buses

Properties

float	depth	`1.0`
float	feedback	`0.7`
float	range_max_hz	`1600.0`

float	range_min_hz	`440.0`
float	rate_hz	`0.5`

Property Descriptions

float **depth** = `1.0`

- void **set_depth**(value: float)
- float **get_depth**()

Governs how high the filter frequencies sweep. Low value will primarily affect bass frequencies. High value can sweep high into the treble. Value can range from 0.1 to 4.

float **feedback** = `0.7`

- void **set_feedback**(value: float)
- float **get_feedback**()

Output percent of modified sound. Value can range from 0.1 to 0.9.

float **range_max_hz** = `1600.0`

- void **set_range_max_hz**(value: float)
- float **get_range_max_hz**()

Determines the maximum frequency affected by the LFO modulations, in Hz. Value can range from 10 to 10000.

float **range_min_hz** = `440.0` 🔗

- void **set_range_min_hz**(value: float)
- float **get_range_min_hz**()

Determines the minimum frequency affected by the LFO modulations, in Hz. Value can range from 10 to 10000.

---

float **rate_hz** = `0.5` 🔗

- void **set_rate_hz**(value: float)
- float **get_rate_hz**()

Adjusts the rate in Hz at which the effect sweeps up and down across the frequency range.

AudioEffectPitchShift

Inherits: AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a pitch-shifting audio effect to an audio bus.

Raises or lowers the pitch of original sound.

Description

Allows modulation of pitch independently of tempo. All frequencies can be increased/decreased with minimal effect on transients.

Tutorials

- Audio buses

Properties

FFTSize	fft_size	`3`
int	oversampling	`4`
float	pitch_scale	`1.0`

Enumerations

enum **FFTSize**: 🔗

FFTSize **FFT_SIZE_256** = `0`

Use a buffer of 256 samples for the Fast Fourier transform. Lowest latency, but least stable over time.

FFTSize **FFT_SIZE_512** = `1`

Use a buffer of 512 samples for the Fast Fourier transform. Low latency, but less stable over time.

FFTSize **FFT_SIZE_1024** = `2`

Use a buffer of 1024 samples for the Fast Fourier transform. This is a compromise between latency and stability over time.

FFTSize **FFT_SIZE_2048** = `3`

Use a buffer of 2048 samples for the Fast Fourier transform. High latency, but stable over time.

FFTSize **FFT_SIZE_4096** = `4`

Use a buffer of 4096 samples for the Fast Fourier transform. Highest latency, but most stable over time.

FFTSize **FFT_SIZE_MAX** = `5`

Represents the size of the FFTSize enum.

Property Descriptions

FFTSize **fft_size** = `3` 🔗

- void **set_fft_size**(value: FFTSize)

- FFTSize **get_fft_size**()

The size of the Fast Fourier transform [https://en.wikipedia.org/wiki/Fast_Fourier_transform] buffer. Higher values smooth out the effect over time, but have greater latency. The effects of this higher latency are especially noticeable on sounds that have sudden amplitude changes.

---

int **oversampling** = `4` 🔗

- void **set_oversampling**(value: int)
- int **get_oversampling**()

The oversampling factor to use. Higher values result in better quality, but are more demanding on the CPU and may cause audio cracking if the CPU can't keep up.

---

float **pitch_scale** = `1.0` 🔗

- void **set_pitch_scale**(value: float)
- float **get_pitch_scale**()

The pitch scale to use. `1.0` is the default pitch and plays sounds unaffected. pitch_scale can range from `0.0` (infinitely low pitch, inaudible) to `16` (16 times higher than the initial pitch).

AudioEffectRecord

Inherits: AudioEffect **<** Resource **<** RefCounted **<** Object

Audio effect used for recording the sound from an audio bus.

Description

Allows the user to record the sound from an audio bus into an AudioStreamWAV. When used on the "Master" audio bus, this includes all audio output by Godot.

Unlike AudioEffectCapture, this effect encodes the recording with the given format (8-bit, 16-bit, or compressed) instead of giving access to the raw audio samples.

Can be used (with an AudioStreamMicrophone) to record from a microphone.

Note: ProjectSettings.audio/driver/enable_input must be `true` for audio input to work. See also that setting's description for caveats related to permissions and operating system privacy settings.

Tutorials

- Recording with microphone
- Audio Microphone Record Demo [https://godotengine.org/asset-library/asset/2760]

Properties

Format	format	`1`

Methods

AudioStreamWAV	get_recording() const
bool	is_recording_active() const
void	set_recording_active(record: bool)

Property Descriptions

Format **format** = `1`

- void **set_format**(value: Format)
- Format **get_format**()

Specifies the format in which the sample will be recorded. See Format for available formats.

Method Descriptions

AudioStreamWAV **get_recording**() const

Returns the recorded sample.

bool **is_recording_active**() const 🔗

Returns whether the recording is active or not.

---

void **set_recording_active**(record: bool) 🔗

If `true`, the sound will be recorded. Note that restarting the recording will remove the previously recorded sample.

AudioEffectReverb

Inherits: AudioEffect **<** Resource **<** RefCounted **<** Object

Adds a reverberation audio effect to an Audio bus.

Description

Simulates the sound of acoustic environments such as rooms, concert halls, caverns, or an open spaces.

Tutorials

- Audio buses
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

float	damping	`0.5`
float	dry	`1.0`
float	hipass	`0.0`
float	predelay_feedback	`0.4`

float	predelay_msec	`150.0`
float	room_size	`0.8`
float	spread	`1.0`
float	wet	`0.5`

Property Descriptions

float **damping** = `0.5` 🔗

- void **set_damping**(value: float)
- float **get_damping**()

Defines how reflective the imaginary room's walls are. Value can range from 0 to 1.

float **dry** = `1.0` 🔗

- void **set_dry**(value: float)
- float **get_dry**()

Output percent of original sound. At 0, only modified sound is outputted. Value can range from 0 to 1.

float **hipass** = `0.0` 🔗

- void **set_hpf**(value: float)
- float **get_hpf**()

High-pass filter passes signals with a frequency higher than a certain cutoff frequency and attenuates signals with frequencies lower than the cutoff frequency. Value can range from 0 to 1.

---

float **predelay_feedback** = `0.4`

- void **set_predelay_feedback**(value: float)
- float **get_predelay_feedback**()

Output percent of predelay. Value can range from 0 to 1.

---

float **predelay_msec** = `150.0`

- void **set_predelay_msec**(value: float)
- float **get_predelay_msec**()

Time between the original signal and the early reflections of the reverb signal, in milliseconds.

---

float **room_size** = `0.8`

- void **set_room_size**(value: float)
- float **get_room_size**()

Dimensions of simulated room. Bigger means more echoes. Value can range from 0 to 1.

---

float **spread** = `1.0`

- void **set_spread**(value: float)

- float **get_spread**()

Widens or narrows the stereo image of the reverb tail. 1 means fully widens. Value can range from 0 to 1.

---

float **wet** = `0.5` 🔗

- void **set_wet**(value: float)
- float **get_wet**()

Output percent of modified sound. At 0, only original sound is outputted. Value can range from 0 to 1.

AudioEffectSpectrumAnalyzer

Inherits: AudioEffect **<** Resource **<** RefCounted **<** Object

Audio effect that can be used for real-time audio visualizations.

Description

This audio effect does not affect sound output, but can be used for real-time audio visualizations.

This resource configures an AudioEffectSpectrumAnalyzerInstance, which performs the actual analysis at runtime. An instance can be acquired with AudioServer.get_bus_effect_instance.

See also AudioStreamGenerator for procedurally generating sounds.

Tutorials

- Audio Spectrum Visualizer Demo [https://godotengine.org/asset-library/asset/2762]

Properties

float	buffer_length	`2.0`
FFTSize	fft_size	`2`

float	tap_back_pos	`0.01`

Enumerations

enum **FFTSize**: 🔗

FFTSize **FFT_SIZE_256** = `0`

Use a buffer of 256 samples for the Fast Fourier transform. Lowest latency, but least stable over time.

FFTSize **FFT_SIZE_512** = `1`

Use a buffer of 512 samples for the Fast Fourier transform. Low latency, but less stable over time.

FFTSize **FFT_SIZE_1024** = `2`

Use a buffer of 1024 samples for the Fast Fourier transform. This is a compromise between latency and stability over time.

FFTSize **FFT_SIZE_2048** = `3`

Use a buffer of 2048 samples for the Fast Fourier transform. High latency, but stable over time.

FFTSize **FFT_SIZE_4096** = `4`

Use a buffer of 4096 samples for the Fast Fourier transform. Highest latency, but most stable over time.

FFTSize **FFT_SIZE_MAX** = `5`

Represents the size of the FFTSize enum.

Property Descriptions

float **buffer_length** = `2.0`

- void **set_buffer_length**(value: float)
- float **get_buffer_length**()

The length of the buffer to keep (in seconds). Higher values keep data around for longer, but require more memory.

FFTSize **fft_size** = `2`

- void **set_fft_size**(value: FFTSize)
- FFTSize **get_fft_size**()

The size of the Fast Fourier transform [https://en.wikipedia.org/wiki/Fast_Fourier_transform] buffer. Higher values smooth out the spectrum analysis over time, but have greater latency. The effects of this higher latency are especially noticeable with sudden amplitude changes.

---

float **tap_back_pos** = `0.01`

- void **set_tap_back_pos**(value: float)
- float **get_tap_back_pos**()

There is currently no description for this property. Please help us by contributing one!

AudioEffectStereoEnhance

Inherits: AudioEffect **<** Resource **<** RefCounted **<** Object

An audio effect that can be used to adjust the intensity of stereo panning.

Description

An audio effect that can be used to adjust the intensity of stereo panning.

Tutorials

- Audio buses

Properties

float	pan_pullout	`1.0`
float	surround	`0.0`
float	time_pullout_ms	`0.0`

Property Descriptions

float **pan_pullout** = `1.0` 🔗

- void **set_pan_pullout**(value: float)
- float **get_pan_pullout**()

Values greater than 1.0 increase intensity of any panning on audio passing through this effect, whereas values less than 1.0 will decrease the panning intensity. A value of 0.0 will downmix audio to mono.

---

float **surround** = `0.0` 🔗

- void **set_surround**(value: float)
- float **get_surround**()

There is currently no description for this property. Please help us by contributing one!

---

float **time_pullout_ms** = `0.0` 🔗

- void **set_time_pullout**(value: float)
- float **get_time_pullout**()

There is currently no description for this property. Please help us by contributing one!

AudioStream

Inherits: Resource **<** RefCounted **<** Object

Inherited By: AudioStreamGenerator, AudioStreamInteractive, AudioStreamMicrophone, AudioStreamMP3, AudioStreamOggVorbis, AudioStreamPlaylist, AudioStreamPolyphonic, AudioStreamRandomizer, AudioStreamSynchronized, AudioStreamWAV

Base class for audio streams.

Description

Base class for audio streams. Audio streams are used for sound effects and music playback, and support WAV (via AudioStreamWAV) and Ogg (via AudioStreamOggVorbis) file formats.

Tutorials

- Audio streams
- Audio Generator Demo [https://godotengine.org/asset-library/asset/2759]
- Audio Microphone Record Demo [https://godotengine.org/asset-library/asset/2760]
- Audio Spectrum Visualizer Demo [https://godotengine.org/asset-library/asset/2762]

Methods

int	_get_beat_count() virtual const
float	_get_bpm() virtual const
float	_get_length() virtual const
Array[Dictionary]	_get_parameter_list() virtual const
String	_get_stream_name() virtual const
AudioStreamPlayback	_instantiate_playback() virtual const
bool	_is_monophonic() virtual const
bool	can_be_sampled() const
AudioSample	generate_sample() const
float	get_length() const
AudioStreamPlayback	instantiate_playback()
bool	is_meta_stream() const
bool	is_monophonic() const

Signals

parameter_list_changed()

Signal to be emitted to notify when the parameter list changed.

Method Descriptions

int **_get_beat_count**() virtual const

Overridable method. Should return the total number of beats of this audio stream. Used by the engine to determine the position of every beat.

Ideally, the returned value should be based off the stream's sample rate (AudioStreamWAV.mix_rate, for example).

---

float **_get_bpm**() virtual const

Overridable method. Should return the tempo of this audio stream, in beats per minute (BPM). Used by the engine to determine the position of every beat.

Ideally, the returned value should be based off the stream's sample rate (AudioStreamWAV.mix_rate, for example).

---

float **_get_length**() virtual const

Override this method to customize the returned value of get_length. Should return the length of this audio stream, in seconds.

Array[Dictionary] **_get_parameter_list**() virtual const 🔗

Return the controllable parameters of this stream. This array contains dictionaries with a property info description format (see Object.get_property_list). Additionally, the default value for this parameter must be added tho each dictionary in "default_value" field.

String **_get_stream_name**() virtual const 🔗

Override this method to customize the name assigned to this audio stream. Unused by the engine.

AudioStreamPlayback **_instantiate_playback**() virtual const 🔗

Override this method to customize the returned value of instantiate_playback. Should returned a new AudioStreamPlayback created when the stream is played (such as by an AudioStreamPlayer)..

---

bool **_is_monophonic**() virtual const 🔗

Override this method to customize the returned value of is_monophonic. Should return `true` if this audio stream only supports one channel.

---

bool **can_be_sampled**() const 🔗

Experimental: This method may be changed or removed in future versions.

Returns if the current **AudioStream** can be used as a sample. Only static streams can be sampled.

AudioSample **generate_sample**() const 🔗

Experimental: This method may be changed or removed in future versions.

Generates an AudioSample based on the current stream.

---

float **get_length**() const 🔗

Returns the length of the audio stream in seconds.

AudioStreamPlayback **instantiate_playback**() 🔗

Returns a newly created AudioStreamPlayback intended to play this audio stream. Useful for when you want to extend _instantiate_playback but call instantiate_playback from an internally held AudioStream subresource. An example of this can be found in the source code for `AudioStreamRandomPitch::instantiate_playback`.

---

bool **is_meta_stream**() const 🔗

Returns `true` if the stream is a collection of other streams, `false` otherwise.

---

bool **is_monophonic**() const 🔗

Returns `true` if this audio stream only supports one channel (*monophony*), or `false` if the audio stream supports two or more channels (*polyphony*).

AudioStreamGenerator

Inherits: AudioStream **<** Resource **<** RefCounted **<** Object

An audio stream with utilities for procedural sound generation.

Description

AudioStreamGenerator is a type of audio stream that does not play back sounds on its own; instead, it expects a script to generate audio data for it. See also AudioStreamGeneratorPlayback.

Here's a sample on how to use it to generate a sine wave:

GDScript

```
var playback # Will hold the AudioStreamGeneratorPlayback.
@onready var sample_hz = $AudioStreamPlayer.stream.mix_rate
var pulse_hz = 440.0 # The frequency of the sound wave.

func _ready():
    $AudioStreamPlayer.play()
    playback = $AudioStreamPlayer.get_stream_playback()
    fill_buffer()

func fill_buffer():
    var phase = 0.0
    var increment = pulse_hz / sample_hz
    var frames_available = playback.get_frames_available()

    for i in range(frames_available):
        playback.push_frame(Vector2.ONE * sin(phase * TAU))
        phase = fmod(phase + increment, 1.0)
```

C#

```
[Export] public AudioStreamPlayer Player { get; set; }

private AudioStreamGeneratorPlayback _playback; // Will hold
the AudioStreamGeneratorPlayback.
private float _sampleHz;
private float _pulseHz = 440.0f; // The frequency of the sound
wave.

public override void _Ready()
{
    if (Player.Stream is AudioStreamGenerator generator) //
Type as a generator to access MixRate.
    {
        _sampleHz = generator.MixRate;
        Player.Play();
        _playback =
(AudioStreamGeneratorPlayback)Player.GetStreamPlayback();
        FillBuffer();
    }
}

public void FillBuffer()
{
    double phase = 0.0;
    float increment = _pulseHz / _sampleHz;
    int framesAvailable = _playback.GetFramesAvailable();

    for (int i = 0; i < framesAvailable; i++)
    {
        _playback.PushFrame(Vector2.One *
(float)Mathf.Sin(phase * Mathf.Tau));
        phase = Mathf.PosMod(phase + increment, 1.0);
    }
}
```

In the example above, the "AudioStreamPlayer" node must use an **AudioStreamGenerator** as its stream. The `fill_buffer` function provides audio data for approximating a sine wave.

See also AudioEffectSpectrumAnalyzer for performing real-time audio spectrum analysis.

Note: Due to performance constraints, this class is best used from C# or from a compiled language via GDExtension. If you still want to use this class from GDScript, consider using a lower mix_rate such as 11,025 Hz or 22,050 Hz.

Tutorials

- Audio Generator Demo [https://godotengine.org/asset-library/asset/2759]

Properties

float	buffer_length	`0.5`
float	mix_rate	`44100.0`

Property Descriptions

float **buffer_length** = `0.5` 🔗

- void **set_buffer_length**(value: float)
- float **get_buffer_length**()

The length of the buffer to generate (in seconds). Lower values result in less latency, but require the script to generate audio data faster, resulting in increased CPU usage and more risk for audio cracking if the CPU can't keep up.

---

float **mix_rate** = `44100.0` 🔗

- void **set_mix_rate**(value: float)
- float **get_mix_rate**()

The sample rate to use (in Hz). Higher values are more demanding for the CPU to generate, but result in better quality.

In games, common sample rates in use are `11025`, `16000`, `22050`, `32000`, `44100`, and `48000`.

According to the Nyquist-Shannon sampling theorem [https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem], there is no quality difference to human hearing when going past 40,000 Hz (since most humans can only hear up to ~20,000 Hz, often less). If you are generating lower-pitched sounds such as voices, lower sample rates such as `32000` or `22050` may be usable with no loss in quality.

AudioStreamInteractive

Inherits: AudioStream **<** Resource **<** RefCounted **<** Object

Audio stream that can playback music interactively, combining clips and a transition table.

Description

This is an audio stream that can playback music interactively, combining clips and a transition table. Clips must be added first, and the transition rules via the add_transition. Additionally, this stream export a property parameter to control the playback via AudioStreamPlayer, AudioStreamPlayer2D, or AudioStreamPlayer3D.

The way this is used is by filling a number of clips, then configuring the transition table. From there, clips are selected for playback and the music will smoothly go from the current to the new one while using the corresponding transition rule defined in the transition table.

Properties

int	clip_count	0
int	initial_clip	0

Methods

void	add_transition(from_clip: int, to_clip: int, from_time: TransitionFromTime, to_time: TransitionToTime, fade_mode: FadeMode, fade_beats: float, use_filler_clip: bool = false, filler_clip: int = -1, hold_previous: bool = false)
void	erase_transition(from_clip: int, to_clip: int)
AutoAdvanceMode	get_clip_auto_advance(clip_index: int) const
int	get_clip_auto_advance_next_clip(clip_index: int) const
StringName	get_clip_name(clip_index: int) const
AudioStream	get_clip_stream(clip_index: int) const
float	get_transition_fade_beats(from_clip: int, to_clip: int) const
FadeMode	get_transition_fade_mode(from_clip: int, to_clip: int) const
int	get_transition_filler_clip(from_clip: int, to_clip: int) const

TransitionFromTime	get_transition_from_time(from_clip: int, to_clip: int) const
PackedInt32Array	get_transition_list() const
TransitionToTime	get_transition_to_time(from_clip: int, to_clip: int) const
bool	has_transition(from_clip: int, to_clip: int) const
bool	is_transition_holding_previous(from_clip: int, to_clip: int) const
bool	is_transition_using_filler_clip(from_clip: int, to_clip: int) const
void	set_clip_auto_advance(clip_index: int, mode: AutoAdvanceMode)
void	set_clip_auto_advance_next_clip(clip_index: int, auto_advance_next_clip: int)
void	set_clip_name(clip_index: int, name: StringName)

void	set_clip_stream(clip_index: int, stream: AudioStream)

Enumerations

enum **TransitionFromTime**: 🔗

TransitionFromTime **TRANSITION_FROM_TIME_IMMEDIATE** = `0`

Start transition as soon as possible, don't wait for any specific time position.

TransitionFromTime **TRANSITION_FROM_TIME_NEXT_BEAT** = `1`

Transition when the clip playback position reaches the next beat.

TransitionFromTime **TRANSITION_FROM_TIME_NEXT_BAR** = `2`

Transition when the clip playback position reaches the next bar.

TransitionFromTime **TRANSITION_FROM_TIME_END** = `3`

Transition when the current clip finished playing.

enum **TransitionToTime**: 🔗

TransitionToTime **TRANSITION_TO_TIME_SAME_POSITION** = `0`

Transition to the same position in the destination clip. This is useful when both clips have exactly the same length and the music should fade between them.

TransitionToTime **TRANSITION_TO_TIME_START** = `1`

Transition to the start of the destination clip.

---

enum **FadeMode**: 🔗

FadeMode **FADE_DISABLED** = `0`

Do not use fade for the transition. This is useful when transitioning from a clip-end to clip-beginning, and each clip has their begin/end.

FadeMode **FADE_IN** = `1`

Use a fade-in in the next clip, let the current clip finish.

FadeMode **FADE_OUT** = `2`

Use a fade-out in the current clip, the next clip will start by itself.

FadeMode **FADE_CROSS** = `3`

Use a cross-fade between clips.

FadeMode **FADE_AUTOMATIC** = `4`

Use automatic fade logic depending on the transition from/to. It is recommended to use this by default.

---

enum **AutoAdvanceMode**: 🔗

AutoAdvanceMode **AUTO_ADVANCE_DISABLED** = `0`

Disable auto-advance (default).

AutoAdvanceMode **AUTO_ADVANCE_ENABLED** = `1`

Enable auto-advance, a clip must be specified.

AutoAdvanceMode **AUTO_ADVANCE_RETURN_TO_HOLD** = `2`

Enable auto-advance, but instead of specifying a clip, the playback will return to hold (see add_transition).

Constants

CLIP_ANY = `-1`

This constant describes that any clip is valid for a specific transition as either source or destination.

Property Descriptions

int **clip_count** = `0`

- void **set_clip_count**(value: int)
- int **get_clip_count**()

Amount of clips contained in this interactive player.

---

int **initial_clip** = `0`

- void **set_initial_clip**(value: int)
- int **get_initial_clip**()

Index of the initial clip, which will be played first when this stream is played.

Method Descriptions

void **add_transition**(from_clip: int, to_clip: int, from_time: TransitionFromTime, to_time: TransitionToTime, fade_mode: FadeMode, fade_beats: float, use_filler_clip: bool = false, filler_clip: int = -1, hold_previous: bool = false) 🔗

Add a transition between two clips. Provide the indices of the source and destination clips, or use the CLIP_ANY constant to indicate that transition happens to/from any clip to this one.

* `from_time` indicates the moment in the current clip the transition will begin after triggered.

* `to_time` indicates the time in the next clip that the playback will start from.

* `fade_mode` indicates how the fade will happen between clips. If unsure, just use FADE_AUTOMATIC which uses the most common type of fade for each situation.

* `fade_beats` indicates how many beats the fade will take. Using decimals is allowed.

* `use_filler_clip` indicates that there will be a filler clip used between the source and destination clips.

* `filler_clip` the index of the filler clip.

* If `hold_previous` is used, then this clip will be remembered. This can be used together with AUTO_ADVANCE_RETURN_TO_HOLD to return to this clip after another is done playing.

---

void **erase_transition**(from_clip: int, to_clip: int)

Erase a transition by providing `from_clip` and `to_clip` clip indices. CLIP_ANY can be used for either argument or both.

AutoAdvanceMode **get_clip_auto_advance**(clip_index: int) const

Return whether a clip has auto-advance enabled. See set_clip_auto_advance.

---

int **get_clip_auto_advance_next_clip**(clip_index: int) const

Return the clip towards which the clip referenced by `clip_index` will auto-advance to.

StringName **get_clip_name**(clip_index: int) const

Return the name of a clip.

AudioStream **get_clip_stream**(clip_index: int) const

Return the AudioStream associated with a clip.

---

float **get_transition_fade_beats**(from_clip: int, to_clip: int) const

Return the time (in beats) for a transition (see add_transition).

FadeMode **get_transition_fade_mode**(from_clip: int, to_clip: int) const

Return the mode for a transition (see add_transition).

int **get_transition_filler_clip**(from_clip: int, to_clip: int) const

Return the filler clip for a transition (see add_transition).

TransitionFromTime **get_transition_from_time**(from_clip: int, to_clip: int) const

Return the source time position for a transition (see add_transition).

PackedInt32Array **get_transition_list**() const

Return the list of transitions (from, to interleaved).

TransitionToTime **get_transition_to_time**(from_clip: int, to_clip: int) const

Return the destination time position for a transition (see add_transition).

bool **has_transition**(from_clip: int, to_clip: int) const

Return true if a given transition exists (was added via add_transition).

bool **is_transition_holding_previous**(from_clip: int, to_clip: int) const

Return whether a transition uses the *hold previous* functionality (see add_transition).

---

bool **is_transition_using_filler_clip**(from_clip: int, to_clip: int) const

Return whether a transition uses the *filler clip* functionality (see add_transition).

---

void **set_clip_auto_advance**(clip_index: int, mode: AutoAdvanceMode)

Set whether a clip will auto-advance by changing the auto-advance mode.

---

void **set_clip_auto_advance_next_clip**(clip_index: int, auto_advance_next_clip: int)

Set the index of the next clip towards which this clip will auto advance to when finished. If the clip being played loops, then auto-advance will be ignored.

---

void **set_clip_name**(clip_index: int, name: StringName)

Set the name of the current clip (for easier identification).

---

void **set_clip_stream**(clip_index: int, stream: AudioStream)

Set the AudioStream associated with the current clip.

AudioStreamMicrophone

Inherits: AudioStream **<** Resource **<** RefCounted **<** Object

Plays real-time audio input data.

Description

When used directly in an AudioStreamPlayer node, **AudioStreamMicrophone** plays back microphone input in real-time. This can be used in conjunction with AudioEffectCapture to process the data or save it.

Note: ProjectSettings.audio/driver/enable_input must be `true` for audio input to work. See also that setting's description for caveats related to permissions and operating system privacy settings.

Tutorials

- Recording with microphone
- Audio Mic Record Demo [https://github.com/godotengine/godot-demo-projects/tree/master/audio/mic_record]

AudioStreamMP3

Inherits: AudioStream **<** Resource **<** RefCounted **<** Object

MP3 audio stream driver.

Description

MP3 audio stream driver. See data if you want to load an MP3 file at run-time.

Properties

int	bar_beats	`4`
int	beat_count	`0`
float	bpm	`0.0`
PackedByteArray	data	`PackedByteArray()`
bool	loop	`false`
float	loop_offset	`0.0`

Property Descriptions

int **bar_beats** = `4` 🔗

- void **set_bar_beats**(value: int)
- int **get_bar_beats**()

There is currently no description for this property. Please help us by contributing one!

---

int **beat_count** = `0` 🔗

- void **set_beat_count**(value: int)
- int **get_beat_count**()

There is currently no description for this property. Please help us by contributing one!

---

float **bpm** = `0.0` 🔗

- void **set_bpm**(value: float)
- float **get_bpm**()

There is currently no description for this property. Please help us by contributing one!

PackedByteArray **data** = `PackedByteArray()` 🔗

- void **set_data**(value: PackedByteArray)
- PackedByteArray **get_data**()

Contains the audio data in bytes.

You can load a file without having to import it beforehand using the code snippet below. Keep in mind that this snippet loads the whole file into memory and may not be ideal for huge files (hundreds of megabytes or more).

GDScript

```
func load_mp3(path):
    var file = FileAccess.open(path, FileAccess.READ)
    var sound = AudioStreamMP3.new()
    sound.data = file.get_buffer(file.get_length())
    return sound
```

C#

```
public AudioStreamMP3 LoadMP3(string path)
{
    using var file = FileAccess.Open(path,
FileAccess.ModeFlags.Read);
    var sound = new AudioStreamMP3();
    sound.Data = file.GetBuffer(file.GetLength());
    return sound;
}
```

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedByteArray for more details.

---

bool **loop** = `false` 🔗

- void **set_loop**(value: bool)
- bool **has_loop**()

If `true`, the stream will automatically loop when it reaches the end.

---

float **loop_offset** = `0.0` 🔗

- void **set_loop_offset**(value: float)
- float **get_loop_offset**()

Time in seconds at which the stream starts after being looped.

AudioStreamOggVorbis

Inherits: AudioStream **<** Resource **<** RefCounted **<** Object

A class representing an Ogg Vorbis audio stream.

Description

The AudioStreamOggVorbis class is a specialized AudioStream for handling Ogg Vorbis file formats. It offers functionality for loading and playing back Ogg Vorbis files, as well as managing looping and other playback properties. This class is part of the audio stream system, which also supports WAV files through the AudioStreamWAV class.

Tutorials

- Runtime file loading and saving

Properties

int	bar_beats	`4`
int	beat_count	`0`
float	bpm	`0.0`

bool	loop	`false`
float	loop_offset	`0.0`
OggPacketSequence	packet_sequence	

Methods

AudioStreamOggVorbis	load_from_buffer(buffer: PackedByteArray) static
AudioStreamOggVorbis	load_from_file(path: String) static

Property Descriptions

int **bar_beats** = `4` 🔗

- void **set_bar_beats**(value: int)
- int **get_bar_beats**()

There is currently no description for this property. Please help us by contributing one!

---

int **beat_count** = `0` 🔗

- void **set_beat_count**(value: int)

- int **get_beat_count**()

There is currently no description for this property. Please help us by contributing one!

---

float **bpm** = `0.0` 🔗

- void **set_bpm**(value: float)
- float **get_bpm**()

There is currently no description for this property. Please help us by contributing one!

---

bool **loop** = `false` 🔗

- void **set_loop**(value: bool)
- bool **has_loop**()

If `true`, the audio will play again from the specified loop_offset once it is done playing. Useful for ambient sounds and background music.

---

float **loop_offset** = `0.0` 🔗

- void **set_loop_offset**(value: float)
- float **get_loop_offset**()

Time in seconds at which the stream starts after being looped.

OggPacketSequence **packet_sequence** 🔗

- void **set_packet_sequence**(value: OggPacketSequence)
- OggPacketSequence **get_packet_sequence**()

Contains the raw Ogg data for this stream.

Method Descriptions

AudioStreamOggVorbis **load_from_buffer**(buffer: PackedByteArray) static 🔗

Creates a new AudioStreamOggVorbis instance from the given buffer. The buffer must contain Ogg Vorbis data.

AudioStreamOggVorbis **load_from_file**(path: String) static 🔗

Creates a new AudioStreamOggVorbis instance from the given file path. The file must be in Ogg Vorbis format.

AudioStreamPlaylist

Inherits: AudioStream **<** Resource **<** RefCounted **<** Object

AudioStream that includes sub-streams and plays them back like a playlist.

Properties

float	fade_time	`0.3`
bool	loop	`true`
bool	shuffle	`false`
int	stream_count	`0`

Methods

float	get_bpm() const
AudioStream	get_list_stream(stream_index: int) const
void	set_list_stream(stream_index: int, audio_stream: AudioStream)

Constants

MAX_STREAMS = `64`

Maximum amount of streams supported in the playlist.

Property Descriptions

float **fade_time** = `0.3`

- void **set_fade_time**(value: float)
- float **get_fade_time**()

Fade time used when a stream ends, when going to the next one. Streams are expected to have an extra bit of audio after the end to help with fading.

bool **loop** = `true`

- void **set_loop**(value: bool)
- bool **has_loop**()

If `true`, the playlist will loop, otherwise the playlist will end when the last stream is finished.

bool **shuffle** = `false`

- void **set_shuffle**(value: bool)
- bool **get_shuffle**()

If `true`, the playlist will shuffle each time playback starts and each time it loops.

---

int **stream_count** = `0`

- void **set_stream_count**(value: int)
- int **get_stream_count**()

Amount of streams in the playlist.

Method Descriptions

float **get_bpm**() const

Returns the BPM of the playlist, which can vary depending on the clip being played.

AudioStream **get_list_stream**(stream_index: int) const

Returns the stream at playback position index.

---

void **set_list_stream**(stream_index: int, audio_stream: AudioStream)

Sets the stream at playback position index.

AudioStreamPolyphonic

Inherits: AudioStream **<** Resource **<** RefCounted **<** Object

AudioStream that lets the user play custom streams at any time from code, simultaneously using a single player.

Description

AudioStream that lets the user play custom streams at any time from code, simultaneously using a single player.

Playback control is done via the AudioStreamPlaybackPolyphonic instance set inside the player, which can be obtained via AudioStreamPlayer.get_stream_playback, AudioStreamPlayer2D.get_stream_playback or AudioStreamPlayer3D.get_stream_playback methods. Obtaining the playback instance is only valid after the `stream` property is set as an **AudioStreamPolyphonic** in those players.

Properties

int	polyphony	`32`

Property Descriptions

int **polyphony** = `32` 🔗

- void **set_polyphony**(value: int)
- int **get_polyphony**()

Maximum amount of simultaneous streams that can be played.

AudioStreamRandomizer

Inherits: AudioStream **<** Resource **<** RefCounted **<** Object

Wraps a pool of audio streams with pitch and volume shifting.

Description

Picks a random AudioStream from the pool, depending on the playback mode, and applies random pitch shifting and volume shifting during playback.

Properties

PlaybackMode	playback_mode	`0`
float	random_pitch	`1.0`
float	random_volume_offset_db	`0.0`
int	streams_count	`0`

Methods

void	add_stream(index: int, stream: AudioStream, weight: float = 1.0)
AudioStream	get_stream(index: int) const
float	get_stream_probability_weight(index: int) const
void	move_stream(index_from: int, index_to: int)
void	remove_stream(index: int)
void	set_stream(index: int, stream: AudioStream)
void	set_stream_probability_weight(index: int, weight: float)

Enumerations

enum **PlaybackMode**: 🔗

PlaybackMode **PLAYBACK_RANDOM_NO_REPEATS** = `0`

Pick a stream at random according to the probability weights chosen for each stream, but avoid playing the same stream twice in a row whenever possible. If only 1 sound is present in the pool, the same sound will always play, effectively allowing repeats to occur.

PlaybackMode **PLAYBACK_RANDOM** = `1`

Pick a stream at random according to the probability weights chosen for each stream. If only 1 sound is present in the pool, the same sound will always play.

PlaybackMode **PLAYBACK_SEQUENTIAL** = `2`

Play streams in the order they appear in the stream pool. If only 1 sound is present in the pool, the same sound will always play.

Property Descriptions

PlaybackMode **playback_mode** = `0`

- void **set_playback_mode**(value: PlaybackMode)
- PlaybackMode **get_playback_mode**()

Controls how this AudioStreamRandomizer picks which AudioStream to play next.

---

float **random_pitch** = `1.0`

- void **set_random_pitch**(value: float)
- float **get_random_pitch**()

The intensity of random pitch variation. A value of 1 means no variation.

---

float **random_volume_offset_db** = `0.0`

- void **set_random_volume_offset_db**(value: float)
- float **get_random_volume_offset_db**()

The intensity of random volume variation. A value of 0 means no variation.

---

int **streams_count** = 0

- void **set_streams_count**(value: int)
- int **get_streams_count**()

The number of streams in the stream pool.

Method Descriptions

void **add_stream**(index: int, stream: AudioStream, weight: float = 1.0)

Insert a stream at the specified index. If the index is less than zero, the insertion occurs at the end of the underlying pool.

AudioStream **get_stream**(index: int) const

Returns the stream at the specified index.

---

float **get_stream_probability_weight**(index: int) const

Returns the probability weight associated with the stream at the given index.

---

void **move_stream**(index_from: int, index_to: int)

Move a stream from one index to another.

---

void **remove_stream**(index: int)

Remove the stream at the specified index.

void **set_stream**(index: int, stream: AudioStream)

Set the AudioStream at the specified index.

void **set_stream_probability_weight**(index: int, weight: float)

Set the probability weight of the stream at the specified index. The higher this value, the more likely that the randomizer will choose this stream during random playback modes.

AudioStreamSynchronized

Inherits: AudioStream **<** Resource **<** RefCounted **<** Object

Stream that can be fitted with sub-streams, which will be played in-sync.

Description

This is a stream that can be fitted with sub-streams, which will be played in-sync. The streams being at exactly the same time when play is pressed, and will end when the last of them ends. If one of the sub-streams loops, then playback will continue.

Properties

int	stream_count	`0`

Methods

AudioStream	get_sync_stream(stream_index: int) const
float	get_sync_stream_volume(stream_index: int) const
void	set_sync_stream(stream_index: int, audio_stream: AudioStream)

void	set_sync_stream_volume(stream_index: int, volume_db: float)

Constants

MAX_STREAMS = `32`

Maximum amount of streams that can be synchronized.

Property Descriptions

int **stream_count** = `0`

- void **set_stream_count**(value: int)
- int **get_stream_count**()

Set the total amount of streams that will be played back synchronized.

Method Descriptions

AudioStream **get_sync_stream**(stream_index: int) const

Get one of the synchronized streams, by index.

---

float **get_sync_stream_volume**(stream_index: int) const

Get the volume of one of the synchronized streams, by index.

---

void **set_sync_stream**(stream_index: int, audio_stream: AudioStream) 🔗

Set one of the synchronized streams, by index.

---

void **set_sync_stream_volume**(stream_index: int, volume_db: float) 🔗

Set the volume of one of the synchronized streams, by index.

AudioStreamWAV

Inherits: AudioStream **<** Resource **<** RefCounted **<** Object

Stores audio data loaded from WAV files.

Description

AudioStreamWAV stores sound samples loaded from WAV files. To play the stored sound, use an AudioStreamPlayer (for non-positional audio) or AudioStreamPlayer2D/AudioStreamPlayer3D (for positional audio). The sound can be looped.

This class can also be used to store dynamically-generated PCM audio data. See also AudioStreamGenerator for procedural audio generation.

Tutorials

- Runtime file loading and saving

Properties

PackedByteArray	data	`PackedByteArray()`
Format	format	`0`
int	loop_begin	`0`

int	loop_end	`0`
LoopMode	loop_mode	`0`
int	mix_rate	`44100`
bool	stereo	`false`

Methods

Error	save_to_wav(path: String)

Enumerations

enum **Format**: 🔗

Format **FORMAT_8_BITS** = `0`

8-bit audio codec.

Format **FORMAT_16_BITS** = `1`

16-bit audio codec.

Format **FORMAT_IMA_ADPCM** = `2`

Audio is compressed using IMA ADPCM.

[Format](#) **FORMAT_QOA** = `3`

Audio is compressed as QOA ([Quite OK Audio](https://qoaformat.org/) [https://qoaformat.org/]).

---

enum **LoopMode**: 🔗

[LoopMode](#) **LOOP_DISABLED** = `0`

Audio does not loop.

[LoopMode](#) **LOOP_FORWARD** = `1`

Audio loops the data between [loop_begin](#) and [loop_end](#), playing forward only.

[LoopMode](#) **LOOP_PINGPONG** = `2`

Audio loops the data between [loop_begin](#) and [loop_end](#), playing back and forth.

[LoopMode](#) **LOOP_BACKWARD** = `3`

Audio loops the data between [loop_begin](#) and [loop_end](#), playing backward only.

Property Descriptions

[PackedByteArray](#) **data** = `PackedByteArray()` 🔗

- void **set_data**(value: [PackedByteArray](#))
- [PackedByteArray](#) **get_data**()

Contains the audio data in bytes.

Note: This property expects signed PCM8 data. To convert unsigned PCM8 to signed PCM8, subtract 128 from each byte.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedByteArray for more details.

Format **format** = `0`

- void **set_format**(value: Format)
- Format **get_format**()

Audio format. See Format constants for values.

int **loop_begin** = `0`

- void **set_loop_begin**(value: int)
- int **get_loop_begin**()

The loop start point (in number of samples, relative to the beginning of the stream). This information will be imported automatically from the WAV file if present.

int **loop_end** = `0`

- void **set_loop_end**(value: int)
- int **get_loop_end**()

The loop end point (in number of samples, relative to the beginning of the stream). This information will be imported automatically from the WAV file if present.

LoopMode **loop_mode** = `0`

- void **set_loop_mode**(value: LoopMode)
- LoopMode **get_loop_mode**()

The loop mode. This information will be imported automatically from the WAV file if present. See LoopMode constants for values.

---

int **mix_rate** = `44100`

- void **set_mix_rate**(value: int)
- int **get_mix_rate**()

The sample rate for mixing this audio. Higher values require more storage space, but result in better quality.

In games, common sample rates in use are `11025`, `16000`, `22050`, `32000`, `44100`, and `48000`.

According to the Nyquist-Shannon sampling theorem [https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem], there is no quality difference to human hearing when going past 40,000 Hz (since most humans can only hear up to ~20,000 Hz, often less). If you are using lower-pitched sounds such as voices, lower sample rates such as `32000` or `22050` may be usable with no loss in quality.

---

bool **stereo** = `false`

- void **set_stereo**(value: bool)
- bool **is_stereo**()

If `true`, audio is stereo.

Method Descriptions

Error **save_to_wav**(path: String) 🔗

Saves the AudioStreamWAV as a WAV file to `path`. Samples with IMA ADPCM or QOA formats can't be saved.

Note: A `.wav` extension is automatically appended to `path` if it is missing.

BaseMaterial3D

Inherits: Material **<** Resource **<** RefCounted **<** Object

Inherited By: ORMMaterial3D, StandardMaterial3D

Abstract base class for defining the 3D rendering properties of meshes.

Description

This class serves as a default material with a wide variety of rendering features and properties without the need to write shader code. See the tutorial below for details.

Tutorials

- Standard Material 3D and ORM Material 3D

Properties

Color	albedo_color	`Color(1, 1, 1, 1)`
Texture2D	albedo_texture	
bool	albedo_texture_force_srgb	`false`
bool	albedo_texture_msdf	`false`

float	alpha_antialiasing_edge	
AlphaAntiAliasing	alpha_antialiasing_mode	
float	alpha_hash_scale	
float	alpha_scissor_threshold	
float	anisotropy	`0.0`
bool	anisotropy_enabled	`false`
Texture2D	anisotropy_flowmap	
bool	ao_enabled	`false`
float	ao_light_affect	`0.0`
bool	ao_on_uv2	`false`
Texture2D	ao_texture	
TextureChannel	ao_texture_channel	`0`
Color	backlight	`Color(0, 0, 0, 1)`

bool	backlight_enabled	`false`
Texture2D	backlight_texture	
bool	billboard_keep_scale	`false`
BillboardMode	billboard_mode	`0`
BlendMode	blend_mode	`0`
float	clearcoat	`1.0`
bool	clearcoat_enabled	`false`
float	clearcoat_roughness	`0.5`
Texture2D	clearcoat_texture	
CullMode	cull_mode	`0`
DepthDrawMode	depth_draw_mode	`0`
Texture2D	detail_albedo	
BlendMode	detail_blend_mode	`0`

bool	detail_enabled	`false`
Texture2D	detail_mask	
Texture2D	detail_normal	
DetailUV	detail_uv_layer	`0`
DiffuseMode	diffuse_mode	`0`
bool	disable_ambient_light	`false`
bool	disable_fog	`false`
bool	disable_receive_shadows	`false`
float	distance_fade_max_distance	`10.0`
float	distance_fade_min_distance	`0.0`
DistanceFadeMode	distance_fade_mode	`0`
Color	emission	`Color(0, 0, 0, 1)`
bool	emission_enabled	`false`

float	emission_energy_multiplier	`1.0`
float	emission_intensity	
bool	emission_on_uv2	`false`
EmissionOperator	emission_operator	`0`
Texture2D	emission_texture	
bool	fixed_size	`false`
bool	grow	`false`
float	grow_amount	`0.0`
bool	heightmap_deep_parallax	`false`
bool	heightmap_enabled	`false`
bool	heightmap_flip_binormal	`false`
bool	heightmap_flip_tangent	`false`
bool	heightmap_flip_texture	`false`

int	heightmap_max_layers	
int	heightmap_min_layers	
float	heightmap_scale	`5.0`
Texture2D	heightmap_texture	
float	metallic	`0.0`
float	metallic_specular	`0.5`
Texture2D	metallic_texture	
TextureChannel	metallic_texture_channel	`0`
float	msdf_outline_size	`0.0`
float	msdf_pixel_range	`4.0`
bool	no_depth_test	`false`
bool	normal_enabled	`false`
float	normal_scale	`1.0`

Texture2D	normal_texture	
Texture2D	orm_texture	
int	particles_anim_h_frames	
bool	particles_anim_loop	
int	particles_anim_v_frames	
float	point_size	`1.0`
float	proximity_fade_distance	`1.0`
bool	proximity_fade_enabled	`false`
bool	refraction_enabled	`false`
float	refraction_scale	`0.05`
Texture2D	refraction_texture	
TextureChannel	refraction_texture_channel	`0`
float	rim	`1.0`

bool	rim_enabled	`false`
Texture2D	rim_texture	
float	rim_tint	`0.5`
float	roughness	`1.0`
Texture2D	roughness_texture	
TextureChannel	roughness_texture_channel	`0`
ShadingMode	shading_mode	`1`
bool	shadow_to_opacity	`false`
SpecularMode	specular_mode	`0`
bool	subsurf_scatter_enabled	`false`
bool	subsurf_scatter_skin_mode	`false`
float	subsurf_scatter_strength	`0.0`
Texture2D	subsurf_scatter_texture	

float	subsurf_scatter_transmittance_boost	`0.0`
Color	subsurf_scatter_transmittance_color	`Color(1, 1, 1, 1)`
float	subsurf_scatter_transmittance_depth	`0.1`
bool	subsurf_scatter_transmittance_enabled	`false`
Texture2D	subsurf_scatter_transmittance_texture	
TextureFilter	texture_filter	`3`
bool	texture_repeat	`true`
Transparency	transparency	`0`
bool	use_particle_trails	`false`
bool	use_point_size	`false`
Vector3	uv1_offset	`Vector3(0, 0, 0)`
Vector3	uv1_scale	`Vector3(1, 1, 1)`

bool	uv1_triplanar	`false`
float	uv1_triplanar_sharpness	`1.0`
bool	uv1_world_triplanar	`false`
Vector3	uv2_offset	`Vector3(0, 0, 0)`
Vector3	uv2_scale	`Vector3(1, 1, 1)`
bool	uv2_triplanar	`false`
float	uv2_triplanar_sharpness	`1.0`
bool	uv2_world_triplanar	`false`
bool	vertex_color_is_srgb	`false`
bool	vertex_color_use_as_albedo	`false`

Methods

bool	get_feature(feature: Feature) const

bool	get_flag(flag: Flags) const
Texture2D	get_texture(param: TextureParam) const
void	set_feature(feature: Feature, enable: bool)
void	set_flag(flag: Flags, enable: bool)
void	set_texture(param: TextureParam, texture: Texture2D)

Enumerations

enum **TextureParam**: 🔗

TextureParam **TEXTURE_ALBEDO** = `0`

Texture specifying per-pixel color.

TextureParam **TEXTURE_METALLIC** = `1`

Texture specifying per-pixel metallic value.

TextureParam **TEXTURE_ROUGHNESS** = `2`

Texture specifying per-pixel roughness value.

TextureParam **TEXTURE_EMISSION** = `3`

Texture specifying per-pixel emission color.

TextureParam **TEXTURE_NORMAL** = `4`

Texture specifying per-pixel normal vector.

TextureParam **TEXTURE_RIM** = `5`

Texture specifying per-pixel rim value.

TextureParam **TEXTURE_CLEARCOAT** = `6`

Texture specifying per-pixel clearcoat value.

TextureParam **TEXTURE_FLOWMAP** = `7`

Texture specifying per-pixel flowmap direction for use with anisotropy.

TextureParam **TEXTURE_AMBIENT_OCCLUSION** = `8`

Texture specifying per-pixel ambient occlusion value.

TextureParam **TEXTURE_HEIGHTMAP** = `9`

Texture specifying per-pixel height.

TextureParam **TEXTURE_SUBSURFACE_SCATTERING** = `10`

Texture specifying per-pixel subsurface scattering.

TextureParam **TEXTURE_SUBSURFACE_TRANSMITTANCE** = `11`

Texture specifying per-pixel transmittance for subsurface scattering.

TextureParam **TEXTURE_BACKLIGHT** = `12`

Texture specifying per-pixel backlight color.

TextureParam **TEXTURE_REFRACTION** = `13`

Texture specifying per-pixel refraction strength.

TextureParam **TEXTURE_DETAIL_MASK** = `14`

Texture specifying per-pixel detail mask blending value.

TextureParam **TEXTURE_DETAIL_ALBEDO** = `15`

Texture specifying per-pixel detail color.

TextureParam **TEXTURE_DETAIL_NORMAL** = `16`

Texture specifying per-pixel detail normal.

TextureParam **TEXTURE_ORM** = `17`

Texture holding ambient occlusion, roughness, and metallic.

TextureParam **TEXTURE_MAX** = `18`

Represents the size of the TextureParam enum.

---

enum **TextureFilter**: 🔗

TextureFilter **TEXTURE_FILTER_NEAREST** = `0`

The texture filter reads from the nearest pixel only. This makes the texture look pixelated from up close, and grainy from a distance (due to mipmaps not being sampled).

TextureFilter **TEXTURE_FILTER_LINEAR** = `1`

The texture filter blends between the nearest 4 pixels. This makes the texture look smooth from up close, and grainy from a distance (due to mipmaps not being sampled).

TextureFilter **TEXTURE_FILTER_NEAREST_WITH_MIPMAPS** = `2`

The texture filter reads from the nearest pixel and blends between the nearest 2 mipmaps (or uses the nearest mipmap if ProjectSettings.rendering/textures/default_filters/use_nearest_mipmap_ filter is `true`). This makes the texture look pixelated from up close, and smooth from a distance.

TextureFilter **TEXTURE_FILTER_LINEAR_WITH_MIPMAPS** = `3`

The texture filter blends between the nearest 4 pixels and between the nearest 2 mipmaps (or uses the nearest mipmap if ProjectSettings.rendering/textures/default_filters/use_nearest_mipmap_filter is `true`). This makes the texture look smooth from up close, and smooth from a distance.

TextureFilter **TEXTURE_FILTER_NEAREST_WITH_MIPMAPS_ANISOTROPIC** = `4`

The texture filter reads from the nearest pixel and blends between 2 mipmaps (or uses the nearest mipmap if ProjectSettings.rendering/textures/default_filters/use_nearest_mipmap_filter is `true`) based on the angle between the surface and the camera view. This makes the texture look pixelated from up close, and smooth from a distance. Anisotropic filtering improves texture quality on surfaces that are almost in line with the camera, but is slightly slower. The anisotropic filtering level can be changed by adjusting ProjectSettings.rendering/textures/default_filters/anisotropic_filtering_level.

TextureFilter **TEXTURE_FILTER_LINEAR_WITH_MIPMAPS_ANISOTROPIC** = `5`

The texture filter blends between the nearest 4 pixels and blends between 2 mipmaps (or uses the nearest mipmap if ProjectSettings.rendering/textures/default_filters/use_nearest_mipmap_filter is `true`) based on the angle between the surface and the camera view. This makes the texture look smooth from up close, and smooth from a distance. Anisotropic filtering improves texture quality on surfaces that are almost in line with the camera, but is slightly slower. The anisotropic filtering level can be changed by adjusting

ProjectSettings.rendering/textures/default_filters/anisotropic_filtering_level.

TextureFilter **TEXTURE_FILTER_MAX** = `6`

Represents the size of the TextureFilter enum.

---

enum **DetailUV**: 🔗

DetailUV **DETAIL_UV_1** = `0`

Use `UV` with the detail texture.

DetailUV **DETAIL_UV_2** = `1`

Use `UV2` with the detail texture.

---

enum **Transparency**: 🔗

Transparency **TRANSPARENCY_DISABLED** = `0`

The material will not use transparency. This is the fastest to render.

Transparency **TRANSPARENCY_ALPHA** = `1`

The material will use the texture's alpha values for transparency. This is the slowest to render, and disables shadow casting.

Transparency **TRANSPARENCY_ALPHA_SCISSOR** = `2`

The material will cut off all values below a threshold, the rest will remain opaque. The opaque portions will be rendered in the depth prepass. This is faster to render than alpha blending, but slower than opaque rendering. This also supports casting shadows.

Transparency **TRANSPARENCY_ALPHA_HASH** = `3`

The material will cut off all values below a spatially-deterministic threshold, the rest will remain opaque. This is faster to render than alpha blending, but slower than opaque rendering. This also supports casting shadows. Alpha hashing is suited for hair rendering.

Transparency **TRANSPARENCY_ALPHA_DEPTH_PRE_PASS** = `4`

The material will use the texture's alpha value for transparency, but will discard fragments with an alpha of less than 0.99 during the depth prepass and fragments with an alpha less than 0.1 during the shadow pass. This also supports casting shadows.

Transparency **TRANSPARENCY_MAX** = `5`

Represents the size of the Transparency enum.

---

enum **ShadingMode**: 🔗

ShadingMode **SHADING_MODE_UNSHADED** = `0`

The object will not receive shadows. This is the fastest to render, but it disables all interactions with lights.

ShadingMode **SHADING_MODE_PER_PIXEL** = `1`

The object will be shaded per pixel. Useful for realistic shading effects.

ShadingMode **SHADING_MODE_PER_VERTEX** = `2`

The object will be shaded per vertex. Useful when you want cheaper shaders and do not care about visual quality. Not implemented yet (this mode will act like SHADING_MODE_PER_PIXEL).

ShadingMode **SHADING_MODE_MAX** = `3`

Represents the size of the ShadingMode enum.

---

enum **Feature**: 🔗

Feature **FEATURE_EMISSION** = `0`

Constant for setting emission_enabled.

Feature **FEATURE_NORMAL_MAPPING** = `1`

Constant for setting normal_enabled.

Feature **FEATURE_RIM** = `2`

Constant for setting rim_enabled.

Feature **FEATURE_CLEARCOAT** = `3`

Constant for setting clearcoat_enabled.

Feature **FEATURE_ANISOTROPY** = `4`

Constant for setting anisotropy_enabled.

Feature **FEATURE_AMBIENT_OCCLUSION** = `5`

Constant for setting ao_enabled.

Feature **FEATURE_HEIGHT_MAPPING** = `6`

Constant for setting heightmap_enabled.

Feature **FEATURE_SUBSURFACE_SCATTERING** = `7`

Constant for setting subsurf_scatter_enabled.

Feature **FEATURE_SUBSURFACE_TRANSMITTANCE** = `8`

Constant for setting subsurf_scatter_transmittance_enabled.

Feature **FEATURE_BACKLIGHT** = `9`

Constant for setting backlight_enabled.

Feature **FEATURE_REFRACTION** = `10`

Constant for setting refraction_enabled.

Feature **FEATURE_DETAIL** = `11`

Constant for setting detail_enabled.

Feature **FEATURE_MAX** = `12`

Represents the size of the Feature enum.

---

enum **BlendMode**: 🔗

BlendMode **BLEND_MODE_MIX** = `0`

Default blend mode. The color of the object is blended over the background based on the object's alpha value.

BlendMode **BLEND_MODE_ADD** = `1`

The color of the object is added to the background.

BlendMode **BLEND_MODE_SUB** = `2`

The color of the object is subtracted from the background.

BlendMode **BLEND_MODE_MUL** = `3`

The color of the object is multiplied by the background.

BlendMode **BLEND_MODE_PREMULT_ALPHA** = `4`

The color of the object is added to the background and the alpha channel is used to mask out the background. This is effectively a hybrid of the blend mix and add modes, useful for effects like fire where you want the flame to add but the smoke to mix. By default, this works with unshaded materials using premultiplied textures. For shaded materials, use the `PREMUL_ALPHA_FACTOR` built-in so that lighting can be modulated as well.

---

enum **AlphaAntiAliasing**: 🔗

AlphaAntiAliasing **ALPHA_ANTIALIASING_OFF** = `0`

Disables Alpha AntiAliasing for the material.

AlphaAntiAliasing **ALPHA_ANTIALIASING_ALPHA_TO_COVERAGE** = `1`

Enables AlphaToCoverage. Alpha values in the material are passed to the AntiAliasing sample mask.

AlphaAntiAliasing **ALPHA_ANTIALIASING_ALPHA_TO_COVERAGE_AND_TO_ONE** = `2`

Enables AlphaToCoverage and forces all non-zero alpha values to `1`. Alpha values in the material are passed to the AntiAliasing sample mask.

---

enum **DepthDrawMode**: 🔗

DepthDrawMode **DEPTH_DRAW_OPAQUE_ONLY** = `0`

Default depth draw mode. Depth is drawn only for opaque objects during the opaque prepass (if any) and during the opaque pass.

DepthDrawMode **DEPTH_DRAW_ALWAYS** = `1`

Objects will write to depth during the opaque and the transparent passes. Transparent objects that are close to the camera may obscure other transparent objects behind them.

Note: This does not influence whether transparent objects are included in the depth prepass or not. For that, see Transparency.

DepthDrawMode **DEPTH_DRAW_DISABLED** = `2`

Objects will not write their depth to the depth buffer, even during the depth prepass (if enabled).

---

enum **CullMode**: 🔗

CullMode **CULL_BACK** = `0`

Default cull mode. The back of the object is culled when not visible. Back face triangles will be culled when facing the camera. This results in only the front side of triangles being drawn. For closed-surface meshes, this means that only the exterior of the mesh will be visible.

CullMode **CULL_FRONT** = `1`

Front face triangles will be culled when facing the camera. This results in only the back side of triangles being drawn. For closed-surface meshes, this means that the interior of the mesh will be drawn instead of the exterior.

CullMode **CULL_DISABLED** = `2`

No face culling is performed; both the front face and back face will be visible.

---

enum **Flags**: 🔗

Flags **FLAG_DISABLE_DEPTH_TEST** = `0`

Disables the depth test, so this object is drawn on top of all others drawn before it. This puts the object in the transparent draw pass where it is sorted based on distance to camera. Objects drawn after it in the draw order may cover it. This also disables writing to depth.

Flags **FLAG_ALBEDO_FROM_VERTEX_COLOR** = `1`

Set `ALBEDO` to the per-vertex color specified in the mesh.

Flags **FLAG_SRGB_VERTEX_COLOR** = `2`

Vertex colors are considered to be stored in sRGB color space and are converted to linear color space during rendering. See also vertex_color_is_srgb.

Note: Only effective when using the Forward+ and Mobile rendering methods.

Flags **FLAG_USE_POINT_SIZE** = `3`

Uses point size to alter the size of primitive points. Also changes the albedo texture lookup to use `POINT_COORD` instead of `UV`.

Flags **FLAG_FIXED_SIZE** = `4`

Object is scaled by depth so that it always appears the same size on screen.

Flags **FLAG_BILLBOARD_KEEP_SCALE** = `5`

Shader will keep the scale set for the mesh. Otherwise the scale is lost when billboarding. Only applies when billboard_mode is BILLBOARD_ENABLED.

Flags **FLAG_UV1_USE_TRIPLANAR** = `6`

Use triplanar texture lookup for all texture lookups that would normally use `UV`.

Flags **FLAG_UV2_USE_TRIPLANAR** = `7`

Use triplanar texture lookup for all texture lookups that would normally use `UV2`.

Flags **FLAG_UV1_USE_WORLD_TRIPLANAR** = `8`

Use triplanar texture lookup for all texture lookups that would normally use `UV`.

Flags **FLAG_UV2_USE_WORLD_TRIPLANAR** = `9`

Use triplanar texture lookup for all texture lookups that would normally use `UV2`.

Flags **FLAG_AO_ON_UV2** = `10`

Use `UV2` coordinates to look up from the ao_texture.

Flags **FLAG_EMISSION_ON_UV2** = `11`

Use `UV2` coordinates to look up from the emission_texture.

Flags **FLAG_ALBEDO_TEXTURE_FORCE_SRGB** = `12`

Forces the shader to convert albedo from sRGB space to linear space. See also albedo_texture_force_srgb.

Flags **FLAG_DONT_RECEIVE_SHADOWS** = `13`

Disables receiving shadows from other objects.

Flags **FLAG_DISABLE_AMBIENT_LIGHT** = `14`

Disables receiving ambient light.

Flags **FLAG_USE_SHADOW_TO_OPACITY** = `15`

Enables the shadow to opacity feature.

Flags **FLAG_USE_TEXTURE_REPEAT** = `16`

Enables the texture to repeat when UV coordinates are outside the 0-1 range. If using one of the linear filtering modes, this can result in artifacts at the edges of a texture when the sampler filters across the edges of the texture.

Flags **FLAG_INVERT_HEIGHTMAP** = `17`

Invert values read from a depth texture to convert them to height values (heightmap).

Flags **FLAG_SUBSURFACE_MODE_SKIN** = `18`

Enables the skin mode for subsurface scattering which is used to improve the look of subsurface scattering when used for human skin.

Flags **FLAG_PARTICLE_TRAILS_MODE** = `19`

Enables parts of the shader required for GPUParticles3D trails to function. This also requires using a mesh with appropriate skinning, such as RibbonTrailMesh or TubeTrailMesh. Enabling this feature outside of materials used in GPUParticles3D meshes will break material rendering.

Flags **FLAG_ALBEDO_TEXTURE_MSDF** = `20`

Enables multichannel signed distance field rendering shader.

Flags **FLAG_DISABLE_FOG** = `21`

Disables receiving depth-based or volumetric fog.

Flags **FLAG_MAX** = `22`

Represents the size of the Flags enum.

---

enum **DiffuseMode**: 🔗

DiffuseMode **DIFFUSE_BURLEY** = `0`

Default diffuse scattering algorithm.

DiffuseMode **DIFFUSE_LAMBERT** = `1`

Diffuse scattering ignores roughness.

DiffuseMode **DIFFUSE_LAMBERT_WRAP** = `2`

Extends Lambert to cover more than 90 degrees when roughness increases.

DiffuseMode **DIFFUSE_TOON** = `3`

Uses a hard cut for lighting, with smoothing affected by roughness.

enum **SpecularMode**: 🔗

SpecularMode **SPECULAR_SCHLICK_GGX** = `0`

Default specular blob.

SpecularMode **SPECULAR_TOON** = `1`

Toon blob which changes size based on roughness.

SpecularMode **SPECULAR_DISABLED** = `2`

No specular blob. This is slightly faster to render than other specular modes.

enum **BillboardMode**: 🔗

BillboardMode **BILLBOARD_DISABLED** = `0`

Billboard mode is disabled.

BillboardMode **BILLBOARD_ENABLED** = `1`

The object's Z axis will always face the camera.

BillboardMode **BILLBOARD_FIXED_Y** = `2`

The object's X axis will always face the camera.

BillboardMode **BILLBOARD_PARTICLES** = `3`

Used for particle systems when assigned to GPUParticles3D and CPUParticles3D nodes (flipbook animation). Enables `particles_anim_*` properties.

The ParticleProcessMaterial.anim_speed_min or CPUParticles3D.anim_speed_min should also be set to a value bigger than zero for the animation to play.

---

enum **TextureChannel**: 🔗

TextureChannel **TEXTURE_CHANNEL_RED** = `0`

Used to read from the red channel of a texture.

TextureChannel **TEXTURE_CHANNEL_GREEN** = `1`

Used to read from the green channel of a texture.

TextureChannel **TEXTURE_CHANNEL_BLUE** = `2`

Used to read from the blue channel of a texture.

TextureChannel **TEXTURE_CHANNEL_ALPHA** = `3`

Used to read from the alpha channel of a texture.

TextureChannel **TEXTURE_CHANNEL_GRAYSCALE** = `4`

Used to read from the linear (non-perceptual) average of the red, green and blue channels of a texture.

---

enum **EmissionOperator**: 🔗

EmissionOperator **EMISSION_OP_ADD** = `0`

Adds the emission color to the color from the emission texture.

EmissionOperator **EMISSION_OP_MULTIPLY** = `1`

Multiplies the emission color by the color from the emission texture.

---

enum **DistanceFadeMode**: 🔗

DistanceFadeMode **DISTANCE_FADE_DISABLED** = `0`

Do not use distance fade.

DistanceFadeMode **DISTANCE_FADE_PIXEL_ALPHA** = `1`

Smoothly fades the object out based on each pixel's distance from the camera using the alpha channel.

DistanceFadeMode **DISTANCE_FADE_PIXEL_DITHER** = `2`

Smoothly fades the object out based on each pixel's distance from the camera using a dithering approach. Dithering discards pixels based on a set pattern to smoothly fade without enabling transparency. On certain hardware, this can be faster than DISTANCE_FADE_PIXEL_ALPHA.

DistanceFadeMode **DISTANCE_FADE_OBJECT_DITHER** = `3`

Smoothly fades the object out based on the object's distance from the camera using a dithering approach. Dithering discards pixels based on a set pattern to smoothly fade without enabling transparency. On certain hardware, this can be faster than DISTANCE_FADE_PIXEL_ALPHA and DISTANCE_FADE_PIXEL_DITHER.

Property Descriptions

Color **albedo_color** = `Color(1, 1, 1, 1)` 🔗

- void **set_albedo**(value: Color)
- Color **get_albedo**()

The material's base color.

Note: If detail_enabled is `true` and a detail_albedo texture is specified, albedo_color will *not* modulate the detail texture. This can be used to color partial areas of a material by not specifying an albedo texture and using a transparent detail_albedo texture instead.

Texture2D **albedo_texture** 🔗

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

Texture to multiply by albedo_color. Used for basic texturing of objects.

If the texture appears unexpectedly too dark or too bright, check albedo_texture_force_srgb.

bool **albedo_texture_force_srgb** = `false` 🔗

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, forces a conversion of the albedo_texture from sRGB color space to linear color space. See also vertex_color_is_srgb.

This should only be enabled when needed (typically when using a ViewportTexture as albedo_texture). If albedo_texture_force_srgb is `true` when it shouldn't be, the texture will appear to be too dark. If albedo_texture_force_srgb is `false` when it shouldn't be, the texture will appear to be too bright.

---

bool **albedo_texture_msdf** = `false` 🔗

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

Enables multichannel signed distance field rendering shader. Use msdf_pixel_range and msdf_outline_size to configure MSDF parameters.

---

float **alpha_antialiasing_edge** 🔗

- void **set_alpha_antialiasing_edge**(value: float)
- float **get_alpha_antialiasing_edge**()

Threshold at which antialiasing will be applied on the alpha channel.

AlphaAntiAliasing **alpha_antialiasing_mode** 🔗

- void **set_alpha_antialiasing**(value: AlphaAntiAliasing)
- AlphaAntiAliasing **get_alpha_antialiasing**()

The type of alpha antialiasing to apply. See AlphaAntiAliasing.

---

float **alpha_hash_scale** 🔗

- void **set_alpha_hash_scale**(value: float)
- float **get_alpha_hash_scale**()

The hashing scale for Alpha Hash. Recommended values between `0` and `2`.

---

float **alpha_scissor_threshold** 🔗

- void **set_alpha_scissor_threshold**(value: float)
- float **get_alpha_scissor_threshold**()

Threshold at which the alpha scissor will discard values. Higher values will result in more pixels being discarded. If the material becomes too opaque at a distance, try increasing alpha_scissor_threshold. If the material disappears at a distance, try decreasing alpha_scissor_threshold.

---

float **anisotropy** = `0.0` 🔗

- void **set_anisotropy**(value: float)
- float **get_anisotropy**()

The strength of the anisotropy effect. This is multiplied by anisotropy_flowmap's alpha channel if a texture is defined there and the texture contains an alpha channel.

---

bool **anisotropy_enabled** = `false` 🔗

- void **set_feature**(feature: Feature, enable: bool)
- bool **get_feature**(feature: Feature) const

If `true`, anisotropy is enabled. Anisotropy changes the shape of the specular blob and aligns it to tangent space. This is useful for brushed aluminium and hair reflections.

Note: Mesh tangents are needed for anisotropy to work. If the mesh does not contain tangents, the anisotropy effect will appear broken.

Note: Material anisotropy should not to be confused with anisotropic texture filtering, which can be enabled by setting texture_filter to TEXTURE_FILTER_LINEAR_WITH_MIPMAPS_ANISOTROPIC.

Texture2D **anisotropy_flowmap** 🔗

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

Texture that offsets the tangent map for anisotropy calculations and optionally controls the anisotropy effect (if an alpha channel is present). The flowmap texture is expected to be a derivative map, with the red channel representing distortion on the X axis and green channel representing distortion on the Y axis. Values below 0.5 will result in negative distortion, whereas values above 0.5 will result in positive distortion.

If present, the texture's alpha channel will be used to multiply the strength of the anisotropy effect. Fully opaque pixels will keep the anisotropy effect's original strength while fully transparent pixels will disable the anisotropy effect entirely. The flowmap texture's blue channel is ignored.

---

bool **ao_enabled** = `false` 🔗

- void **set_feature**(feature: Feature, enable: bool)
- bool **get_feature**(feature: Feature) const

If `true`, ambient occlusion is enabled. Ambient occlusion darkens areas based on the ao_texture.

---

float **ao_light_affect** = `0.0` 🔗

- void **set_ao_light_affect**(value: float)
- float **get_ao_light_affect**()

Amount that ambient occlusion affects lighting from lights. If `0`, ambient occlusion only affects ambient light. If `1`, ambient occlusion affects lights just as much as it affects ambient light. This can be used to impact the strength of the ambient occlusion effect, but typically looks unrealistic.

---

bool **ao_on_uv2** = `false`

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, use `UV2` coordinates to look up from the ao_texture.

Texture2D **ao_texture**

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

Texture that defines the amount of ambient occlusion for a given point on the object.

TextureChannel **ao_texture_channel** = `0`

- void **set_ao_texture_channel**(value: TextureChannel)
- TextureChannel **get_ao_texture_channel**()

Specifies the channel of the ao_texture in which the ambient occlusion information is stored. This is useful when you store the information for multiple effects in a single texture. For example if you stored metallic in the red channel, roughness in the blue, and ambient occlusion in the green you could reduce the number of textures you use.

Color **backlight** = `Color(0, 0, 0, 1)`

- void **set_backlight**(value: Color)
- Color **get_backlight**()

The color used by the backlight effect. Represents the light passing through an object.

---

bool **backlight_enabled** = `false`

- void **set_feature**(feature: Feature, enable: bool)
- bool **get_feature**(feature: Feature) const

If `true`, the backlight effect is enabled. See also subsurf_scatter_transmittance_enabled.

Texture2D **backlight_texture**

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

Texture used to control the backlight effect per-pixel. Added to backlight.

---

bool **billboard_keep_scale** = `false`

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, the shader will keep the scale set for the mesh. Otherwise, the scale is lost when billboarding. Only applies when billboard_mode is not BILLBOARD_DISABLED.

BillboardMode **billboard_mode** = `0`

- void **set_billboard_mode**(value: BillboardMode)
- BillboardMode **get_billboard_mode**()

Controls how the object faces the camera. See BillboardMode.

Note: When billboarding is enabled and the material also casts shadows, billboards will face **the** camera in the scene when rendering shadows. In scenes with multiple cameras, the intended shadow cannot be determined and this will result in undefined behavior. See GitHub Pull Request #72638 [https://github.com/godotengine/godot/pull/72638] for details.

Note: Billboard mode is not suitable for VR because the left-right vector of the camera is not horizontal when the screen is attached to your head instead of on the table. See GitHub issue #41567 [https://github.com/godotengine/godot/issues/41567] for details.

BlendMode **blend_mode** = `0`

- void **set_blend_mode**(value: BlendMode)
- BlendMode **get_blend_mode**()

The material's blend mode.

Note: Values other than `Mix` force the object into the transparent pipeline. See BlendMode.

---

float **clearcoat** = `1.0`

- void **set_clearcoat**(value: float)
- float **get_clearcoat**()

Sets the strength of the clearcoat effect. Setting to `0` looks the same as disabling the clearcoat effect.

---

bool **clearcoat_enabled** = `false`

- void **set_feature**(feature: Feature, enable: bool)
- bool **get_feature**(feature: Feature) const

If `true`, clearcoat rendering is enabled. Adds a secondary transparent pass to the lighting calculation resulting in an added specular blob. This makes materials appear as if they have a clear layer on them that can be either glossy or rough.

Note: Clearcoat rendering is not visible if the material's shading_mode is SHADING_MODE_UNSHADED.

---

float **clearcoat_roughness** = `0.5` 🔗

- void **set_clearcoat_roughness**(value: float)
- float **get_clearcoat_roughness**()

Sets the roughness of the clearcoat pass. A higher value results in a rougher clearcoat while a lower value results in a smoother clearcoat.

Texture2D **clearcoat_texture** 🔗

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

Texture that defines the strength of the clearcoat effect and the glossiness of the clearcoat. Strength is specified in the red channel while glossiness is specified in the green channel.

CullMode **cull_mode** = `0` 🔗

- void **set_cull_mode**(value: CullMode)
- CullMode **get_cull_mode**()

Determines which side of the triangle to cull depending on whether the triangle faces towards or away from the camera. See CullMode.

DepthDrawMode **depth_draw_mode** = `0` 🔗

- void **set_depth_draw_mode**(value: DepthDrawMode)
- DepthDrawMode **get_depth_draw_mode**()

Determines when depth rendering takes place. See DepthDrawMode. See also transparency.

Texture2D **detail_albedo** 🔗

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

Texture that specifies the color of the detail overlay. detail_albedo's alpha channel is used as a mask, even when the material is opaque. To use a dedicated texture as a mask, see detail_mask.

Note: detail_albedo is *not* modulated by albedo_color.

BlendMode **detail_blend_mode** = `0`

- void **set_detail_blend_mode**(value: BlendMode)
- BlendMode **get_detail_blend_mode**()

Specifies how the detail_albedo should blend with the current `ALBEDO`. See BlendMode for options.

---

bool **detail_enabled** = `false`

- void **set_feature**(feature: Feature, enable: bool)
- bool **get_feature**(feature: Feature) const

If `true`, enables the detail overlay. Detail is a second texture that gets mixed over the surface of the object based on detail_mask and detail_albedo's alpha channel. This can be used to add variation to objects, or to blend between two different albedo/normal textures.

Texture2D **detail_mask**

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

Texture used to specify how the detail textures get blended with the base textures. detail_mask can be used together with detail_albedo's alpha channel (if any).

Texture2D **detail_normal** 🔗

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

Texture that specifies the per-pixel normal of the detail overlay. The detail_normal texture only uses the red and green channels; the blue and alpha channels are ignored. The normal read from detail_normal is oriented around the surface normal provided by the Mesh.

Note: Godot expects the normal map to use X+, Y+, and Z+ coordinates. See this page [http://wiki.polycount.com/wiki/Normal_Map_Technical_Details#Common_Swizzle_Coordinates] for a comparison of normal map coordinates expected by popular engines.

DetailUV **detail_uv_layer** = `0` 🔗

- void **set_detail_uv**(value: DetailUV)
- DetailUV **get_detail_uv**()

Specifies whether to use `UV` or `UV2` for the detail layer. See DetailUV for options.

DiffuseMode **diffuse_mode** = `0` 🔗

- void **set_diffuse_mode**(value: DiffuseMode)
- DiffuseMode **get_diffuse_mode**()

The algorithm used for diffuse light scattering. See DiffuseMode.

---

bool **disable_ambient_light** = `false` 🔗

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, the object receives no ambient light.

---

bool **disable_fog** = `false`

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, the object will not be affected by fog (neither volumetric nor depth fog). This is useful for unshaded or transparent materials (e.g. particles), which without this setting will be affected even if fully transparent.

---

bool **disable_receive_shadows** = `false`

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, the object receives no shadow that would otherwise be cast onto it.

---

float **distance_fade_max_distance** = `10.0`

- void **set_distance_fade_max_distance**(value: float)
- float **get_distance_fade_max_distance**()

Distance at which the object appears fully opaque.

Note: If distance_fade_max_distance is less than distance_fade_min_distance, the behavior will be reversed. The object will start to fade away at distance_fade_max_distance and will fully disappear once it reaches distance_fade_min_distance.

---

float **distance_fade_min_distance** = `0.0`

- void **set_distance_fade_min_distance**(value: float)

- float **get_distance_fade_min_distance**()

Distance at which the object starts to become visible. If the object is less than this distance away, it will be invisible.

Note: If distance_fade_min_distance is greater than distance_fade_max_distance, the behavior will be reversed. The object will start to fade away at distance_fade_max_distance and will fully disappear once it reaches distance_fade_min_distance.

DistanceFadeMode **distance_fade_mode** = `0` 🔗

- void **set_distance_fade**(value: DistanceFadeMode)
- DistanceFadeMode **get_distance_fade**()

Specifies which type of fade to use. Can be any of the DistanceFadeModes.

Color **emission** = `Color(0, 0, 0, 1)` 🔗

- void **set_emission**(value: Color)
- Color **get_emission**()

The emitted light's color. See emission_enabled.

---

bool **emission_enabled** = `false` 🔗

- void **set_feature**(feature: Feature, enable: bool)
- bool **get_feature**(feature: Feature) const

If `true`, the body emits light. Emitting light makes the object appear brighter. The object can also cast light on other objects if a VoxelGI, SDFGI, or LightmapGI is used and this object is used in baked lighting.

---

float **emission_energy_multiplier** = `1.0` 🔗

- void **set_emission_energy_multiplier**(value: float)
- float **get_emission_energy_multiplier**()

Multiplier for emitted light. See emission_enabled.

---

float **emission_intensity**

- void **set_emission_intensity**(value: float)
- float **get_emission_intensity**()

Luminance of emitted light, measured in nits (candela per square meter). Only available when ProjectSettings.rendering/lights_and_shadows/use_physical_light_units is enabled. The default is roughly equivalent to an indoor lightbulb.

---

bool **emission_on_uv2** = `false`

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

Use `UV2` to read from the emission_texture.

EmissionOperator **emission_operator** = `0`

- void **set_emission_operator**(value: EmissionOperator)
- EmissionOperator **get_emission_operator**()

Sets how emission interacts with emission_texture. Can either add or multiply. See EmissionOperator for options.

Texture2D **emission_texture**

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

Texture that specifies how much surface emits light at a given point.

---

bool **fixed_size** = `false` 🔗

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, the object is rendered at the same size regardless of distance.

---

bool **grow** = `false` 🔗

- void **set_grow_enabled**(value: bool)
- bool **is_grow_enabled**()

If `true`, enables the vertex grow setting. This can be used to create mesh-based outlines using a second material pass and its cull_mode set to CULL_FRONT. See also grow_amount.

Note: Vertex growth cannot create new vertices, which means that visible gaps may occur in sharp corners. This can be alleviated by designing the mesh to use smooth normals exclusively using face weighted normals [https://wiki.polycount.com/wiki/Face_weighted_normals] in the 3D authoring software. In this case, grow will be able to join every outline together, just like in the original mesh.

---

float **grow_amount** = `0.0` 🔗

- void **set_grow**(value: float)
- float **get_grow**()

Grows object vertices in the direction of their normals. Only effective if grow is `true`.

---

bool **heightmap_deep_parallax** = `false` 🔗

- void **set_heightmap_deep_parallax**(value: bool)
- bool **is_heightmap_deep_parallax_enabled**()

If `true`, uses parallax occlusion mapping to represent depth in the material instead of simple offset mapping (see heightmap_enabled). This results in a more convincing depth effect, but is much more expensive on the GPU. Only enable this on materials where it makes a significant visual difference.

---

bool **heightmap_enabled** = `false` 🔗

- void **set_feature**(feature: Feature, enable: bool)
- bool **get_feature**(feature: Feature) const

If `true`, height mapping is enabled (also called "parallax mapping" or "depth mapping"). See also normal_enabled. Height mapping is a demanding feature on the GPU, so it should only be used on materials where it makes a significant visual difference.

Note: Height mapping is not supported if triplanar mapping is used on the same material. The value of heightmap_enabled will be ignored if uv1_triplanar is enabled.

---

bool **heightmap_flip_binormal** = `false` 🔗

- void **set_heightmap_deep_parallax_flip_binormal**(value: bool)
- bool **get_heightmap_deep_parallax_flip_binormal**()

If `true`, flips the mesh's binormal vectors when interpreting the height map. If the heightmap effect looks strange when the camera moves (even with a reasonable heightmap_scale), try setting this to `true`.

---

bool **heightmap_flip_tangent** = `false` 🔗

- void **set_heightmap_deep_parallax_flip_tangent**(value: bool)
- bool **get_heightmap_deep_parallax_flip_tangent**()

If `true`, flips the mesh's tangent vectors when interpreting the height map. If the heightmap effect looks strange when the camera moves (even with a reasonable heightmap_scale), try setting this to `true`.

---

bool **heightmap_flip_texture** = `false` 🔗

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, interprets the height map texture as a depth map, with brighter values appearing to be "lower" in altitude compared to darker values.

This can be enabled for compatibility with some materials authored for Godot 3.x. This is not necessary if the Invert import option was used to invert the depth map in Godot 3.x, in which case heightmap_flip_texture should remain `false`.

---

int **heightmap_max_layers** 🔗

- void **set_heightmap_deep_parallax_max_layers**(value: int)
- int **get_heightmap_deep_parallax_max_layers**()

The number of layers to use for parallax occlusion mapping when the camera is up close to the material. Higher values result in a more convincing depth effect, especially in materials that have steep height changes. Higher values have a significant cost on the GPU, so it should only be increased on materials where it makes a significant visual difference.

Note: Only effective if heightmap_deep_parallax is `true`.

---

int **heightmap_min_layers** 🔗

- void **set_heightmap_deep_parallax_min_layers**(value: int)
- int **get_heightmap_deep_parallax_min_layers**()

The number of layers to use for parallax occlusion mapping when the camera is far away from the material. Higher values result in a more convincing depth effect, especially in materials that have steep height changes. Higher values have a significant cost on the GPU, so it should only be increased on materials where it makes a significant visual difference.

Note: Only effective if heightmap_deep_parallax is `true`.

---

float **heightmap_scale** = `5.0` 🔗

- void **set_heightmap_scale**(value: float)
- float **get_heightmap_scale**()

The heightmap scale to use for the parallax effect (see heightmap_enabled). The default value is tuned so that the highest point (value = 255) appears to be 5 cm higher than the lowest point (value = 0). Higher values result in a deeper appearance, but may result in artifacts appearing when looking at the material from oblique angles, especially when the camera moves. Negative values can be used to invert the parallax effect, but this is different from inverting the texture using heightmap_flip_texture as the material will also appear to be "closer" to the camera. In most cases, heightmap_scale should be kept to a positive value.

Note: If the height map effect looks strange regardless of this value, try adjusting heightmap_flip_binormal and heightmap_flip_tangent. See also heightmap_texture for recommendations on authoring heightmap textures, as the way the heightmap texture is authored affects how heightmap_scale behaves.

Texture2D **heightmap_texture** 🔗

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

The texture to use as a height map. See also heightmap_enabled.

For best results, the texture should be normalized (with heightmap_scale reduced to compensate). In GIMP [https://gimp.org], this can be done using **Colors > Auto > Equalize**. If the texture only uses a small part of its available range, the parallax effect may look strange, especially when the camera moves.

Note: To reduce memory usage and improve loading times, you may be able to use a lower-resolution heightmap texture as most heightmaps are only comprised of low-frequency data.

---

float **metallic** = `0.0`

- void **set_metallic**(value: float)
- float **get_metallic**()

A high value makes the material appear more like a metal. Non-metals use their albedo as the diffuse color and add diffuse to the specular reflection. With non-metals, the reflection appears on top of the albedo color. Metals use their albedo as a multiplier to the specular reflection and set the diffuse color to black resulting in a tinted reflection. Materials work better when fully metal or fully non-metal, values between `0` and `1` should only be used for blending between metal and non-metal sections. To alter the amount of reflection use roughness.

---

float **metallic_specular** = `0.5`

- void **set_specular**(value: float)
- float **get_specular**()

Adjusts the strength of specular reflections. Specular reflections are composed of scene reflections and the specular lobe which is the bright spot that is reflected from light sources. When set to `0.0`, no specular reflections will be visible. This differs from the SPECULAR_DISABLED SpecularMode as SPECULAR_DISABLED only applies to the specular lobe from the light source.

Note: Unlike metallic, this is not energy-conserving, so it should be left at `0.5` in most cases. See also roughness.

Texture2D **metallic_texture**

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

Texture used to specify metallic for an object. This is multiplied by metallic.

TextureChannel **metallic_texture_channel** = `0`

- void **set_metallic_texture_channel**(value: TextureChannel)
- TextureChannel **get_metallic_texture_channel**()

Specifies the channel of the metallic_texture in which the metallic information is stored. This is useful when you store the information for multiple effects in a single texture. For example if you stored metallic in the red channel, roughness in the blue, and ambient occlusion in the green you could reduce the number of textures you use.

float **msdf_outline_size** = `0.0`

- void **set_msdf_outline_size**(value: float)
- float **get_msdf_outline_size**()

The width of the shape outline.

float **msdf_pixel_range** = `4.0` 🔗

- void **set_msdf_pixel_range**(value: float)
- float **get_msdf_pixel_range**()

The width of the range around the shape between the minimum and maximum representable signed distance.

---

bool **no_depth_test** = `false` 🔗

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, depth testing is disabled and the object will be drawn in render order.

---

bool **normal_enabled** = `false` 🔗

- void **set_feature**(feature: Feature, enable: bool)
- bool **get_feature**(feature: Feature) const

If `true`, normal mapping is enabled. This has a slight performance cost, especially on mobile GPUs.

---

float **normal_scale** = `1.0` 🔗

- void **set_normal_scale**(value: float)
- float **get_normal_scale**()

The strength of the normal map's effect.

Texture2D **normal_texture** 🔗

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

Texture used to specify the normal at a given pixel. The normal_texture only uses the red and green channels; the blue and alpha channels are ignored. The normal read from normal_texture is oriented around the surface normal provided by the Mesh.

Note: The mesh must have both normals and tangents defined in its vertex data. Otherwise, the normal map won't render correctly and will only appear to darken the whole surface. If creating geometry with SurfaceTool, you can use SurfaceTool.generate_normals and SurfaceTool.generate_tangents to automatically generate normals and tangents respectively.

Note: Godot expects the normal map to use X+, Y+, and Z+ coordinates. See this page [http://wiki.polycount.com/wiki/Normal_Map_Technical_Details#Common_Swizzle_Coordinates] for a comparison of normal map coordinates expected by popular engines.

Note: If detail_enabled is `true`, the detail_albedo texture is drawn *below* the normal_texture. To display a normal map *above* the detail_albedo texture, use detail_normal instead.

Texture2D **orm_texture** 🔗

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

The Occlusion/Roughness/Metallic texture to use. This is a more efficient replacement of ao_texture, roughness_texture and metallic_texture in ORMMaterial3D. Ambient occlusion is stored in the red channel. Roughness map is stored in the green channel. Metallic map is stored in the blue channel. The alpha channel is ignored.

---

int **particles_anim_h_frames** 🔗

- void **set_particles_anim_h_frames**(value: int)

- int **get_particles_anim_h_frames**()

The number of horizontal frames in the particle sprite sheet. Only enabled when using BILLBOARD_PARTICLES. See billboard_mode.

---

bool **particles_anim_loop**

- void **set_particles_anim_loop**(value: bool)
- bool **get_particles_anim_loop**()

If `true`, particle animations are looped. Only enabled when using BILLBOARD_PARTICLES. See billboard_mode.

---

int **particles_anim_v_frames**

- void **set_particles_anim_v_frames**(value: int)
- int **get_particles_anim_v_frames**()

The number of vertical frames in the particle sprite sheet. Only enabled when using BILLBOARD_PARTICLES. See billboard_mode.

---

float **point_size** = `1.0`

- void **set_point_size**(value: float)
- float **get_point_size**()

The point size in pixels. See use_point_size.

---

float **proximity_fade_distance** = `1.0`

- void **set_proximity_fade_distance**(value: float)
- float **get_proximity_fade_distance**()

Distance over which the fade effect takes place. The larger the distance the longer it takes for an object to fade.

---

bool **proximity_fade_enabled** = `false` 🔗

- void **set_proximity_fade_enabled**(value: bool)
- bool **is_proximity_fade_enabled**()

If `true`, the proximity fade effect is enabled. The proximity fade effect fades out each pixel based on its distance to another object.

---

bool **refraction_enabled** = `false` 🔗

- void **set_feature**(feature: Feature, enable: bool)
- bool **get_feature**(feature: Feature) const

If `true`, the refraction effect is enabled. Distorts transparency based on light from behind the object.

---

float **refraction_scale** = `0.05` 🔗

- void **set_refraction**(value: float)
- float **get_refraction**()

The strength of the refraction effect.

Texture2D **refraction_texture** 🔗

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

Texture that controls the strength of the refraction per-pixel. Multiplied by refraction_scale.

TextureChannel **refraction_texture_channel** = `0` 🔗

- void **set_refraction_texture_channel**(value: TextureChannel)

- TextureChannel **get_refraction_texture_channel**()

Specifies the channel of the refraction_texture in which the refraction information is stored. This is useful when you store the information for multiple effects in a single texture. For example if you stored refraction in the red channel, roughness in the blue, and ambient occlusion in the green you could reduce the number of textures you use.

float **rim** = `1.0`

- void **set_rim**(value: float)
- float **get_rim**()

Sets the strength of the rim lighting effect.

bool **rim_enabled** = `false`

- void **set_feature**(feature: Feature, enable: bool)
- bool **get_feature**(feature: Feature) const

If `true`, rim effect is enabled. Rim lighting increases the brightness at glancing angles on an object.

Note: Rim lighting is not visible if the material's shading_mode is SHADING_MODE_UNSHADED.

Texture2D **rim_texture**

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

Texture used to set the strength of the rim lighting effect per-pixel. Multiplied by rim.

float **rim_tint** = `0.5`

- void **set_rim_tint**(value: float)
- float **get_rim_tint**()

The amount of to blend light and albedo color when rendering rim effect. If `0` the light color is used, while `1` means albedo color is used. An intermediate value generally works best.

float **roughness** = `1.0` 🔗

- void **set_roughness**(value: float)
- float **get_roughness**()

Surface reflection. A value of `0` represents a perfect mirror while a value of `1` completely blurs the reflection. See also metallic.

Texture2D **roughness_texture** 🔗

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

Texture used to control the roughness per-pixel. Multiplied by roughness.

TextureChannel **roughness_texture_channel** = `0` 🔗

- void **set_roughness_texture_channel**(value: TextureChannel)
- TextureChannel **get_roughness_texture_channel**()

Specifies the channel of the roughness_texture in which the roughness information is stored. This is useful when you store the information for multiple effects in a single texture. For example if you stored metallic in the red channel, roughness in the blue, and ambient occlusion in the green you could reduce the number of textures you use.

ShadingMode **shading_mode** = `1` 🔗

- void **set_shading_mode**(value: ShadingMode)
- ShadingMode **get_shading_mode**()

Sets whether the shading takes place, per-pixel, per-vertex or unshaded. Per-vertex lighting is faster, making it the best choice for mobile applications, however it looks considerably worse than per-pixel. Unshaded rendering is the fastest, but disables all interactions with lights.

Note: Setting the shading mode vertex shading currently has no effect, as vertex shading is not implemented yet.

bool **shadow_to_opacity** = `false` 🔗

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, enables the "shadow to opacity" render mode where lighting modifies the alpha so shadowed areas are opaque and non-shadowed areas are transparent. Useful for overlaying shadows onto a camera feed in AR.

SpecularMode **specular_mode** = `0` 🔗

- void **set_specular_mode**(value: SpecularMode)
- SpecularMode **get_specular_mode**()

The method for rendering the specular blob. See SpecularMode.

Note: specular_mode only applies to the specular blob. It does not affect specular reflections from the sky, screen-space reflections, VoxelGI, SDFGI or ReflectionProbes. To disable reflections from these sources as well, set metallic_specular to `0.0` instead.

bool **subsurf_scatter_enabled** = `false`

- void **set_feature**(feature: Feature, enable: bool)
- bool **get_feature**(feature: Feature) const

If `true`, subsurface scattering is enabled. Emulates light that penetrates an object's surface, is scattered, and then emerges. Subsurface scattering quality is controlled by ProjectSettings.rendering/environment/subsurface_scattering/subsurface_scattering_quality.

---

bool **subsurf_scatter_skin_mode** = `false`

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, subsurface scattering will use a special mode optimized for the color and density of human skin, such as boosting the intensity of the red channel in subsurface scattering.

---

float **subsurf_scatter_strength** = `0.0`

- void **set_subsurface_scattering_strength**(value: float)
- float **get_subsurface_scattering_strength**()

The strength of the subsurface scattering effect. The depth of the effect is also controlled by ProjectSettings.rendering/environment/subsurface_scattering/subsurface_scattering_scale, which is set globally.

Texture2D **subsurf_scatter_texture**

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

Texture used to control the subsurface scattering strength. Stored in the red texture channel. Multiplied by subsurf_scatter_strength.

---

float **subsurf_scatter_transmittance_boost** = `0.0` 🔗

- void **set_transmittance_boost**(value: float)
- float **get_transmittance_boost**()

The intensity of the subsurface scattering transmittance effect.

Color **subsurf_scatter_transmittance_color** = `Color(1, 1, 1, 1)` 🔗

- void **set_transmittance_color**(value: Color)
- Color **get_transmittance_color**()

The color to multiply the subsurface scattering transmittance effect with. Ignored if subsurf_scatter_skin_mode is `true`.

---

float **subsurf_scatter_transmittance_depth** = `0.1` 🔗

- void **set_transmittance_depth**(value: float)
- float **get_transmittance_depth**()

The depth of the subsurface scattering transmittance effect.

---

bool **subsurf_scatter_transmittance_enabled** = `false` 🔗

- void **set_feature**(feature: Feature, enable: bool)
- bool **get_feature**(feature: Feature) const

If `true`, enables subsurface scattering transmittance. Only effective if subsurf_scatter_enabled is `true`. See also backlight_enabled.

Texture2D **subsurf_scatter_transmittance_texture** 🔗

- void **set_texture**(param: TextureParam, texture: Texture2D)
- Texture2D **get_texture**(param: TextureParam) const

The texture to use for multiplying the intensity of the subsurface scattering transmittance intensity. See also subsurf_scatter_texture. Ignored if subsurf_scatter_skin_mode is `true`.

TextureFilter **texture_filter** = `3` 🔗

- void **set_texture_filter**(value: TextureFilter)
- TextureFilter **get_texture_filter**()

Filter flags for the texture. See TextureFilter for options.

Note: heightmap_texture is always sampled with linear filtering, even if nearest-neighbor filtering is selected here. This is to ensure the heightmap effect looks as intended. If you need sharper height transitions between pixels, resize the heightmap texture in an image editor with nearest-neighbor filtering.

---

bool **texture_repeat** = `true` 🔗

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

Repeat flags for the texture. See TextureFilter for options.

Transparency **transparency** = `0` 🔗

- void **set_transparency**(value: Transparency)
- Transparency **get_transparency**()

The material's transparency mode. Some transparency modes will disable shadow casting. Any transparency mode other than

TRANSPARENCY_DISABLED has a greater performance impact compared to opaque rendering. See also blend_mode.

---

bool **use_particle_trails** = `false` 🔗

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, enables parts of the shader required for GPUParticles3D trails to function. This also requires using a mesh with appropriate skinning, such as RibbonTrailMesh or TubeTrailMesh. Enabling this feature outside of materials used in GPUParticles3D meshes will break material rendering.

---

bool **use_point_size** = `false` 🔗

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, render point size can be changed.

Note: This is only effective for objects whose geometry is point-based rather than triangle-based. See also point_size.

Vector3 **uv1_offset** = `Vector3(0, 0, 0)` 🔗

- void **set_uv1_offset**(value: Vector3)
- Vector3 **get_uv1_offset**()

How much to offset the `UV` coordinates. This amount will be added to `UV` in the vertex function. This can be used to offset a texture. The Z component is used when uv1_triplanar is enabled, but it is not used anywhere else.

Vector3 **uv1_scale** = `Vector3(1, 1, 1)`

- void **set_uv1_scale**(value: Vector3)
- Vector3 **get_uv1_scale**()

How much to scale the `UV` coordinates. This is multiplied by `UV` in the vertex function. The Z component is used when uv1_triplanar is enabled, but it is not used anywhere else.

---

bool **uv1_triplanar** = `false`

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, instead of using `UV` textures will use a triplanar texture lookup to determine how to apply textures. Triplanar uses the orientation of the object's surface to blend between texture coordinates. It reads from the source texture 3 times, once for each axis and then blends between the results based on how closely the pixel aligns with each axis. This is often used for natural features to get a realistic blend of materials. Because triplanar texturing requires many more texture reads per-pixel it is much slower than normal UV texturing. Additionally, because it is blending the texture between the three axes, it is unsuitable when you are trying to achieve crisp texturing.

---

float **uv1_triplanar_sharpness** = `1.0`

- void **set_uv1_triplanar_blend_sharpness**(value: float)
- float **get_uv1_triplanar_blend_sharpness**()

A lower number blends the texture more softly while a higher number blends the texture more sharply.

Note: uv1_triplanar_sharpness is clamped between `0.0` and `150.0` (inclusive) as values outside that range can look broken depending on the mesh.

---

bool **uv1_world_triplanar** = `false` 🔗

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, triplanar mapping for `UV` is calculated in world space rather than object local space. See also uv1_triplanar.

Vector3 **uv2_offset** = `Vector3(0, 0, 0)` 🔗

- void **set_uv2_offset**(value: Vector3)
- Vector3 **get_uv2_offset**()

How much to offset the `UV2` coordinates. This amount will be added to `UV2` in the vertex function. This can be used to offset a texture. The Z component is used when uv2_triplanar is enabled, but it is not used anywhere else.

Vector3 **uv2_scale** = `Vector3(1, 1, 1)` 🔗

- void **set_uv2_scale**(value: Vector3)
- Vector3 **get_uv2_scale**()

How much to scale the `UV2` coordinates. This is multiplied by `UV2` in the vertex function. The Z component is used when uv2_triplanar is enabled, but it is not used anywhere else.

---

bool **uv2_triplanar** = `false` 🔗

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, instead of using `UV2` textures will use a triplanar texture lookup to determine how to apply textures. Triplanar uses the orientation of the object's surface to blend between texture coordinates. It reads from the source texture 3 times, once for each axis and then blends between the results based on how closely the pixel aligns with each axis. This is often used for natural features to get a realistic blend of materials. Because triplanar texturing requires many more texture reads per-pixel it is much slower than normal UV texturing. Additionally, because it is blending the texture between the three axes, it is unsuitable when you are trying to achieve crisp texturing.

---

float **uv2_triplanar_sharpness** = `1.0`

- void **set_uv2_triplanar_blend_sharpness**(value: float)
- float **get_uv2_triplanar_blend_sharpness**()

A lower number blends the texture more softly while a higher number blends the texture more sharply.

Note: uv2_triplanar_sharpness is clamped between `0.0` and `150.0` (inclusive) as values outside that range can look broken depending on the mesh.

---

bool **uv2_world_triplanar** = `false`

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, triplanar mapping for `UV2` is calculated in world space rather than object local space. See also uv2_triplanar.

---

bool **vertex_color_is_srgb** = `false`

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, vertex colors are considered to be stored in sRGB color space and are converted to linear color space during rendering. If `false`, vertex colors are considered to be stored in linear color space and are rendered as-is. See also albedo_texture_force_srgb.

Note: Only effective when using the Forward+ and Mobile rendering methods, not Compatibility.

---

bool **vertex_color_use_as_albedo** = `false`

- void **set_flag**(flag: Flags, enable: bool)
- bool **get_flag**(flag: Flags) const

If `true`, the vertex color is used as albedo color.

Method Descriptions

bool **get_feature**(feature: Feature) const

Returns `true`, if the specified Feature is enabled.

---

bool **get_flag**(flag: Flags) const

Returns `true`, if the specified flag is enabled. See Flags enumerator for options.

Texture2D **get_texture**(param: TextureParam) const

Returns the Texture2D associated with the specified TextureParam.

---

void **set_feature**(feature: Feature, enable: bool)

If `true`, enables the specified Feature. Many features that are available in **BaseMaterial3D**s need to be enabled before use. This way the cost for using the feature is only incurred when specified. Features can also be enabled by setting the corresponding member to `true`.

---

void **set_flag**(flag: Flags, enable: bool)

If `true`, enables the specified flag. Flags are optional behavior that can be turned on and off. Only one flag can be enabled at a time with this function, the flag enumerators cannot be bit-masked together to enable or disable multiple flags at once. Flags can also be enabled by setting the corresponding member to `true`. See Flags enumerator for options.

---

void **set_texture**(param: TextureParam, texture: Texture2D)

Sets the texture for the slot specified by `param`. See TextureParam for available slots.

BitMap

Inherits: Resource **<** RefCounted **<** Object

Boolean matrix.

Description

A two-dimensional array of boolean values, can be used to efficiently store a binary matrix (every matrix element takes only one bit) and query the values using natural cartesian coordinates.

Methods

Image	convert_to_image() const
void	create(size: Vector2i)
void	create_from_image_alpha(image: Image, threshold: float = 0.1)
bool	get_bit(x: int, y: int) const
bool	get_bitv(position: Vector2i) const
Vector2i	get_size() const

int	get_true_bit_count() const
void	grow_mask(pixels: int, rect: Rect2i)
Array[PackedVector2Array]	opaque_to_polygons(rect: Rect2i, epsilon: float = 2.0) const
void	resize(new_size: Vector2i)
void	set_bit(x: int, y: int, bit: bool)
void	set_bit_rect(rect: Rect2i, bit: bool)
void	set_bitv(position: Vector2i, bit: bool)

Method Descriptions

Image **convert_to_image**() const

Returns an image of the same size as the bitmap and with a Format of type Image.FORMAT_L8. `true` bits of the bitmap are being converted into white pixels, and `false` bits into black.

---

void **create**(size: Vector2i)

Creates a bitmap with the specified size, filled with `false`.

---

void **create_from_image_alpha**(image: Image, threshold: float = 0.1)

Creates a bitmap that matches the given image dimensions, every element of the bitmap is set to `false` if the alpha value of the image at that position is equal to `threshold` or less, and `true` in other case.

---

bool **get_bit**(x: int, y: int) const

Returns bitmap's value at the specified position.

---

bool **get_bitv**(position: Vector2i) const

Returns bitmap's value at the specified position.

Vector2i **get_size**() const

Returns bitmap's dimensions.

---

int **get_true_bit_count**() const

Returns the number of bitmap elements that are set to `true`.

---

void **grow_mask**(pixels: int, rect: Rect2i)

Applies morphological dilation or erosion to the bitmap. If `pixels` is positive, dilation is applied to the bitmap. If `pixels` is negative, erosion is applied to the bitmap. `rect` defines the area where the morphological operation is applied. Pixels located outside the `rect` are unaffected by grow_mask.

Array[PackedVector2Array] **opaque_to_polygons**(rect: Rect2i, epsilon: float = 2.0) const 🔗

Creates an Array of polygons covering a rectangular portion of the bitmap. It uses a marching squares algorithm, followed by Ramer-Douglas-Peucker (RDP) reduction of the number of vertices. Each polygon is described as a PackedVector2Array of its vertices.

To get polygons covering the whole bitmap, pass:

```
Rect2(Vector2(), get_size())
```

`epsilon` is passed to RDP to control how accurately the polygons cover the bitmap: a lower `epsilon` corresponds to more points in the polygons.

---

void **resize**(new_size: Vector2i) 🔗

Resizes the image to `new_size`.

---

void **set_bit**(x: int, y: int, bit: bool) 🔗

Sets the bitmap's element at the specified position, to the specified value.

---

void **set_bit_rect**(rect: Rect2i, bit: bool) 🔗

Sets a rectangular portion of the bitmap to the specified value.

---

void **set_bitv**(position: Vector2i, bit: bool) 🔗

Sets the bitmap's element at the specified position, to the specified value.

BoneMap

Inherits: Resource **<** RefCounted **<** Object

Describes a mapping of bone names for retargeting Skeleton3D into common names defined by a SkeletonProfile.

Description

This class contains a dictionary that uses a list of bone names in SkeletonProfile as key names.

By assigning the actual Skeleton3D bone name as the key value, it maps the Skeleton3D to the SkeletonProfile.

Tutorials

- Retargeting 3D Skeletons

Properties

SkeletonProfile	profile

Methods

StringName	find_profile_bone_name(skeleton_bone_name: StringName) const

StringName	get_skeleton_bone_name(profile_bone_name: StringName) const
void	set_skeleton_bone_name(profile_bone_name: StringName, skeleton_bone_name: StringName)

Signals

bone_map_updated() 🔗

This signal is emitted when change the key value in the **BoneMap**. This is used to validate mapping and to update **BoneMap** editor.

profile_updated() 🔗

This signal is emitted when change the value in profile or change the reference of profile. This is used to update key names in the **BoneMap** and to redraw the **BoneMap** editor.

Property Descriptions

SkeletonProfile **profile** 🔗

- void **set_profile**(value: SkeletonProfile)
- SkeletonProfile **get_profile**()

A SkeletonProfile of the mapping target. Key names in the **BoneMap** are synchronized with it.

BoxMesh

Inherits: PrimitiveMesh **<** Mesh **<** Resource **<** RefCounted **<** Object

Generate an axis-aligned box PrimitiveMesh.

Description

Generate an axis-aligned box PrimitiveMesh.

The box's UV layout is arranged in a 3×2 layout that allows texturing each face individually. To apply the same texture on all faces, change the material's UV property to `Vector3(3, 2, 1)`. This is equivalent to adding `UV *= vec2(3.0, 2.0)` in a vertex shader.

Note: When using a large textured **BoxMesh** (e.g. as a floor), you may stumble upon UV jittering issues depending on the camera angle. To solve this, increase subdivide_depth, subdivide_height and subdivide_width until you no longer notice UV jittering.

Properties

Vector3	size	`Vector3(1, 1, 1)`
int	subdivide_depth	`0`
int	subdivide_height	`0`

int	subdivide_width	`0`

Property Descriptions

Vector3 **size** = `Vector3(1, 1, 1)` 🔗

- void **set_size**(value: Vector3)
- Vector3 **get_size**()

The box's width, height and depth.

int **subdivide_depth** = `0` 🔗

- void **set_subdivide_depth**(value: int)
- int **get_subdivide_depth**()

Number of extra edge loops inserted along the Z axis.

int **subdivide_height** = `0` 🔗

- void **set_subdivide_height**(value: int)
- int **get_subdivide_height**()

Number of extra edge loops inserted along the Y axis.

int **subdivide_width** = `0` 🔗

- void **set_subdivide_width**(value: int)
- int **get_subdivide_width**()

Number of extra edge loops inserted along the X axis.

BoxOccluder3D

Inherits: Occluder3D **<** Resource **<** RefCounted **<** Object

Cuboid shape for use with occlusion culling in OccluderInstance3D.

Description

BoxOccluder3D stores a cuboid shape that can be used by the engine's occlusion culling system.

See OccluderInstance3D's documentation for instructions on setting up occlusion culling.

Tutorials

- Occlusion culling

Properties

Vector3	size	`Vector3(1, 1, 1)`

Property Descriptions

Vector3 **size** = `Vector3(1, 1, 1)` 🔗

- void **set_size**(value: Vector3)

- Vector3 **get_size**()

The box's size in 3D units.

BoxShape3D

Inherits: Shape3D **<** Resource **<** RefCounted **<** Object

A 3D box shape used for physics collision.

Description

A 3D box shape, intended for use in physics. Usually used to provide a shape for a CollisionShape3D.

Performance: BoxShape3D is fast to check collisions against. It is faster than CapsuleShape3D and CylinderShape3D, but slower than SphereShape3D.

Tutorials

- 3D Physics Tests Demo [https://godotengine.org/asset-library/asset/2747]
- 3D Kinematic Character Demo [https://godotengine.org/asset-library/asset/2739]
- 3D Platformer Demo [https://godotengine.org/asset-library/asset/2748]

Properties

Vector3	size	`Vector3(1, 1, 1)`

Property Descriptions

Vector3 **size** = `Vector3(1, 1, 1)` 🔗

- void **set_size**(value: Vector3)
- Vector3 **get_size**()

The box's width, height and depth.

ButtonGroup

Inherits: Resource **<** RefCounted **<** Object

A group of buttons that doesn't allow more than one button to be pressed at a time.

Description

A group of BaseButton-derived buttons. The buttons in a **ButtonGroup** are treated like radio buttons: No more than one button can be pressed at a time. Some types of buttons (such as CheckBox) may have a special appearance in this state.

Every member of a **ButtonGroup** should have BaseButton.toggle_mode set to `true`.

Properties

bool	allow_unpress	`false`
bool	resource_local_to_scene	`true` (overrides Resource)

Methods

Array[BaseButton]	get_buttons()

BaseButton	get_pressed_button()

Signals

pressed(button: BaseButton)

Emitted when one of the buttons of the group is pressed.

Property Descriptions

bool **allow_unpress** = `false`

- void **set_allow_unpress**(value: bool)
- bool **is_allow_unpress**()

If `true`, it is possible to unpress all buttons in this **ButtonGroup**.

Method Descriptions

Array[BaseButton] **get_buttons**()

Returns an Array of Buttons who have this as their **ButtonGroup** (see BaseButton.button_group).

BaseButton **get_pressed_button**()

Returns the current pressed button.

CameraAttributes

Inherits: Resource **<** RefCounted **<** Object

Inherited By: CameraAttributesPhysical, CameraAttributesPractical

Parent class for camera settings.

Description

Controls camera-specific attributes such as depth of field and exposure override.

When used in a WorldEnvironment it provides default settings for exposure, auto-exposure, and depth of field that will be used by all cameras without their own **CameraAttributes**, including the editor camera. When used in a Camera3D it will override any **CameraAttributes** set in the WorldEnvironment. When used in VoxelGI or LightmapGI, only the exposure settings will be used.

See also Environment for general 3D environment settings.

This is a pure virtual class that is inherited by CameraAttributesPhysical and CameraAttributesPractical.

Properties

bool	auto_exposure_enabled	`false`
float	auto_exposure_scale	`0.4`

float	auto_exposure_speed	`0.5`
float	exposure_multiplier	`1.0`
float	exposure_sensitivity	`100.0`

Property Descriptions

bool **auto_exposure_enabled** = `false`

- void **set_auto_exposure_enabled**(value: bool)
- bool **is_auto_exposure_enabled**()

If `true`, enables the tonemapping auto exposure mode of the scene renderer. If `true`, the renderer will automatically determine the exposure setting to adapt to the scene's illumination and the observed light.

---

float **auto_exposure_scale** = `0.4`

- void **set_auto_exposure_scale**(value: float)
- float **get_auto_exposure_scale**()

The scale of the auto exposure effect. Affects the intensity of auto exposure.

---

float **auto_exposure_speed** = `0.5`

- void **set_auto_exposure_speed**(value: float)

- float **get_auto_exposure_speed**()

The speed of the auto exposure effect. Affects the time needed for the camera to perform auto exposure.

---

float **exposure_multiplier** = `1.0`

- void **set_exposure_multiplier**(value: float)
- float **get_exposure_multiplier**()

Multiplier for the exposure amount. A higher value results in a brighter image.

---

float **exposure_sensitivity** = `100.0`

- void **set_exposure_sensitivity**(value: float)
- float **get_exposure_sensitivity**()

Sensitivity of camera sensors, measured in ISO. A higher sensitivity results in a brighter image. Only available when ProjectSettings.rendering/lights_and_shadows/use_physical_light_units is enabled. When auto_exposure_enabled this can be used as a method of exposure compensation, doubling the value will increase the exposure value (measured in EV100) by 1 stop.

CameraAttributesPhysical

Inherits: CameraAttributes **<** Resource **<** RefCounted **<** Object

Physically-based camera settings.

Description

CameraAttributesPhysical is used to set rendering settings based on a physically-based camera's settings. It is responsible for exposure, auto-exposure, and depth of field.

When used in a WorldEnvironment it provides default settings for exposure, auto-exposure, and depth of field that will be used by all cameras without their own CameraAttributes, including the editor camera. When used in a Camera3D it will override any CameraAttributes set in the WorldEnvironment and will override the Camera3Ds Camera3D.far, Camera3D.near, Camera3D.fov, and Camera3D.keep_aspect properties. When used in VoxelGI or LightmapGI, only the exposure settings will be used.

The default settings are intended for use in an outdoor environment, tips for settings for use in an indoor environment can be found in each setting's documentation.

Note: Depth of field blur is only supported in the Forward+ and Mobile rendering methods, not Compatibility.

Tutorials

- Physical light and camera units

Properties

float	auto_exposure_max_exposure_value	`10.0`
float	auto_exposure_min_exposure_value	`-8.0`
float	exposure_aperture	`16.0`
float	exposure_shutter_speed	`100.0`
float	frustum_far	`4000.0`
float	frustum_focal_length	`35.0`
float	frustum_focus_distance	`10.0`
float	frustum_near	`0.05`

Methods

float	get_fov() const

Property Descriptions

[float](#) **auto_exposure_max_exposure_value** = `10.0` [🔗](#)

- void **set_auto_exposure_max_exposure_value**(value: [float](#))
- [float](#) **get_auto_exposure_max_exposure_value**()

The maximum luminance (in EV100) used when calculating auto exposure. When calculating scene average luminance, color values will be clamped to at least this value. This limits the auto-exposure from exposing below a certain brightness, resulting in a cut off point where the scene will remain bright.

[float](#) **auto_exposure_min_exposure_value** = `-8.0` [🔗](#)

- void **set_auto_exposure_min_exposure_value**(value: [float](#))
- [float](#) **get_auto_exposure_min_exposure_value**()

The minimum luminance luminance (in EV100) used when calculating auto exposure. When calculating scene average luminance, color values will be clamped to at least this value. This limits the auto-exposure from exposing above a certain brightness, resulting in a cut off point where the scene will remain dark.

[float](#) **exposure_aperture** = `16.0` [🔗](#)

- void **set_aperture**(value: [float](#))
- [float](#) **get_aperture**()

Size of the aperture of the camera, measured in f-stops. An f-stop is a unitless ratio between the focal length of the camera and the diameter of the aperture. A high aperture setting will result in a smaller aperture which leads to a dimmer image and sharper focus. A low aperture results in a wide aperture which lets in more light resulting in a brighter, less-focused image. Default is appropriate for outdoors at daytime (i.e. for use with a default DirectionalLight3D), for indoor lighting, a value between 2 and 4 is more appropriate.

Only available when ProjectSettings.rendering/lights_and_shadows/use_physical_light_units is enabled.

---

float **exposure_shutter_speed** = `100.0`

- void **set_shutter_speed**(value: float)
- float **get_shutter_speed**()

Time for shutter to open and close, evaluated as `1 / shutter_speed` seconds. A higher value will allow less light (leading to a darker image), while a lower value will allow more light (leading to a brighter image).

Only available when ProjectSettings.rendering/lights_and_shadows/use_physical_light_units is enabled.

---

float **frustum_far** = `4000.0`

- void **set_far**(value: float)
- float **get_far**()

Override value for Camera3D.far. Used internally when calculating depth of field. When attached to a Camera3D as its Camera3D.attributes, it will override the Camera3D.far property.

---

float **frustum_focal_length** = `35.0`

- void **set_focal_length**(value: float)

- float **get_focal_length**()

Distance between camera lens and camera aperture, measured in millimeters. Controls field of view and depth of field. A larger focal length will result in a smaller field of view and a narrower depth of field meaning fewer objects will be in focus. A smaller focal length will result in a wider field of view and a larger depth of field meaning more objects will be in focus. When attached to a Camera3D as its Camera3D.attributes, it will override the Camera3D.fov property and the Camera3D.keep_aspect property.

---

float **frustum_focus_distance** = `10.0`

- void **set_focus_distance**(value: float)
- float **get_focus_distance**()

Distance from camera of object that will be in focus, measured in meters. Internally this will be clamped to be at least 1 millimeter larger than frustum_focal_length.

---

float **frustum_near** = `0.05`

- void **set_near**(value: float)
- float **get_near**()

Override value for Camera3D.near. Used internally when calculating depth of field. When attached to a Camera3D as its Camera3D.attributes, it will override the Camera3D.near property.

Method Descriptions

float **get_fov**() const

Returns the vertical field of view that corresponds to the frustum_focal_length. This value is calculated internally whenever frustum_focal_length is changed.

CameraAttributesPractical

Inherits: CameraAttributes **<** Resource **<** RefCounted **<** Object

Camera settings in an easy to use format.

Description

Controls camera-specific attributes such as auto-exposure, depth of field, and exposure override.

When used in a WorldEnvironment it provides default settings for exposure, auto-exposure, and depth of field that will be used by all cameras without their own CameraAttributes, including the editor camera. When used in a Camera3D it will override any CameraAttributes set in the WorldEnvironment. When used in VoxelGI or LightmapGI, only the exposure settings will be used.

Properties

float	auto_exposure_max_sensitivity	`800.0`
float	auto_exposure_min_sensitivity	`0.0`
float	dof_blur_amount	`0.1`
float	dof_blur_far_distance	`10.0`

bool	dof_blur_far_enabled	`false`
float	dof_blur_far_transition	`5.0`
float	dof_blur_near_distance	`2.0`
bool	dof_blur_near_enabled	`false`
float	dof_blur_near_transition	`1.0`

Property Descriptions

float **auto_exposure_max_sensitivity** = `800.0` 🔗

- void **set_auto_exposure_max_sensitivity**(value: float)
- float **get_auto_exposure_max_sensitivity**()

The maximum sensitivity (in ISO) used when calculating auto exposure. When calculating scene average luminance, color values will be clamped to at least this value. This limits the auto-exposure from exposing below a certain brightness, resulting in a cut off point where the scene will remain bright.

---

float **auto_exposure_min_sensitivity** = `0.0` 🔗

- void **set_auto_exposure_min_sensitivity**(value: float)
- float **get_auto_exposure_min_sensitivity**()

The minimum sensitivity (in ISO) used when calculating auto exposure. When calculating scene average luminance, color values will be clamped to at least this value. This limits the auto-exposure from exposing above a certain brightness, resulting in a cut off point where the scene will remain dark.

---

float **dof_blur_amount** = `0.1` 🔗

- void **set_dof_blur_amount**(value: float)
- float **get_dof_blur_amount**()

Sets the maximum amount of blur. When using physically-based blur amounts, will instead act as a multiplier. High values lead to an increased amount of blurriness, but can be much more expensive to calculate. It is best to keep this as low as possible for a given art style.

---

float **dof_blur_far_distance** = `10.0` 🔗

- void **set_dof_blur_far_distance**(value: float)
- float **get_dof_blur_far_distance**()

Objects further from the Camera3D by this amount will be blurred by the depth of field effect. Measured in meters.

---

bool **dof_blur_far_enabled** = `false` 🔗

- void **set_dof_blur_far_enabled**(value: bool)
- bool **is_dof_blur_far_enabled**()

Enables depth of field blur for objects further than dof_blur_far_distance. Strength of blur is controlled by dof_blur_amount and modulated by dof_blur_far_transition.

Note: Depth of field blur is only supported in the Forward+ and Mobile rendering methods, not Compatibility.

---

float **dof_blur_far_transition** = `5.0`

- void **set_dof_blur_far_transition**(value: float)
- float **get_dof_blur_far_transition**()

When positive, distance over which (starting from dof_blur_far_distance) blur effect will scale from 0 to dof_blur_amount. When negative, uses physically-based scaling so depth of field effect will scale from 0 at dof_blur_far_distance and will increase in a physically accurate way as objects get further from the Camera3D.

---

float **dof_blur_near_distance** = `2.0`

- void **set_dof_blur_near_distance**(value: float)
- float **get_dof_blur_near_distance**()

Objects closer from the Camera3D by this amount will be blurred by the depth of field effect. Measured in meters.

---

bool **dof_blur_near_enabled** = `false`

- void **set_dof_blur_near_enabled**(value: bool)
- bool **is_dof_blur_near_enabled**()

Enables depth of field blur for objects closer than dof_blur_near_distance. Strength of blur is controlled by dof_blur_amount and modulated by dof_blur_near_transition.

Note: Depth of field blur is only supported in the Forward+ and Mobile rendering methods, not Compatibility.

---

float **dof_blur_near_transition** = `1.0` 🔗

- void **set_dof_blur_near_transition**(value: float)
- float **get_dof_blur_near_transition**()

When positive, distance over which blur effect will scale from 0 to dof_blur_amount, ending at dof_blur_near_distance. When negative, uses physically-based scaling so depth of field effect will scale from 0 at dof_blur_near_distance and will increase in a physically accurate way as objects get closer to the Camera3D.

CameraTexture

Inherits: Texture2D **<** Texture **<** Resource **<** RefCounted **<** Object

Texture provided by a CameraFeed.

Description

This texture gives access to the camera texture provided by a CameraFeed.

Note: Many cameras supply YCbCr images which need to be converted in a shader.

Properties

int	camera_feed_id	`0`
bool	camera_is_active	`false`
bool	resource_local_to_scene	`false` (overrides Resource)
FeedImage	which_feed	`0`

Property Descriptions

int **camera_feed_id** = `0` 🔗

- void **set_camera_feed_id**(value: int)
- int **get_camera_feed_id**()

The ID of the CameraFeed for which we want to display the image.

---

bool **camera_is_active** = `false` 🔗

- void **set_camera_active**(value: bool)
- bool **get_camera_active**()

Convenience property that gives access to the active property of the CameraFeed.

FeedImage **which_feed** = `0` 🔗

- void **set_which_feed**(value: FeedImage)
- FeedImage **get_which_feed**()

Which image within the CameraFeed we want access to, important if the camera image is split in a Y and CbCr component.

CanvasItemMaterial

Inherits: Material **<** Resource **<** RefCounted **<** Object

A material for CanvasItems.

Description

CanvasItemMaterials provide a means of modifying the textures associated with a CanvasItem. They specialize in describing blend and lighting behaviors for textures. Use a ShaderMaterial to more fully customize a material's interactions with a CanvasItem.

Properties

BlendMode	blend_mode	`0`
LightMode	light_mode	`0`
int	particles_anim_h_frames	
bool	particles_anim_loop	
int	particles_anim_v_frames	
bool	particles_animation	`false`

Enumerations

enum **BlendMode**:

BlendMode **BLEND_MODE_MIX** = `0`

Mix blending mode. Colors are assumed to be independent of the alpha (opacity) value.

BlendMode **BLEND_MODE_ADD** = `1`

Additive blending mode.

BlendMode **BLEND_MODE_SUB** = `2`

Subtractive blending mode.

BlendMode **BLEND_MODE_MUL** = `3`

Multiplicative blending mode.

BlendMode **BLEND_MODE_PREMULT_ALPHA** = `4`

Mix blending mode. Colors are assumed to be premultiplied by the alpha (opacity) value.

---

enum **LightMode**:

LightMode **LIGHT_MODE_NORMAL** = `0`

Render the material using both light and non-light sensitive material properties.

LightMode **LIGHT_MODE_UNSHADED** = `1`

Render the material as if there were no light.

LightMode **LIGHT_MODE_LIGHT_ONLY** = `2`

Render the material as if there were only light.

Property Descriptions

BlendMode **blend_mode** = `0` 🔗

- void **set_blend_mode**(value: BlendMode)
- BlendMode **get_blend_mode**()

The manner in which a material's rendering is applied to underlying textures.

LightMode **light_mode** = `0` 🔗

- void **set_light_mode**(value: LightMode)
- LightMode **get_light_mode**()

The manner in which material reacts to lighting.

---

int **particles_anim_h_frames** 🔗

- void **set_particles_anim_h_frames**(value: int)
- int **get_particles_anim_h_frames**()

The number of columns in the spritesheet assigned as Texture2D for a GPUParticles2D or CPUParticles2D.

Note: This property is only used and visible in the editor if particles_animation is `true`.

---

bool **particles_anim_loop**

- void **set_particles_anim_loop**(value: bool)
- bool **get_particles_anim_loop**()

If `true`, the particles animation will loop.

Note: This property is only used and visible in the editor if particles_animation is `true`.

---

int **particles_anim_v_frames**

- void **set_particles_anim_v_frames**(value: int)
- int **get_particles_anim_v_frames**()

The number of rows in the spritesheet assigned as Texture2D for a GPUParticles2D or CPUParticles2D.

Note: This property is only used and visible in the editor if particles_animation is `true`.

---

bool **particles_animation** = `false`

- void **set_particles_animation**(value: bool)
- bool **get_particles_animation**()

If `true`, enable spritesheet-based animation features when assigned to GPUParticles2D and CPUParticles2D nodes. The ParticleProcessMaterial.anim_speed_max or CPUParticles2D.anim_speed_max should also be set to a positive value for the animation to play.

This property (and other `particles_anim_*` properties that depend on it) has no effect on other types of nodes.

CanvasTexture

Inherits: Texture2D **<** Texture **<** Resource **<** RefCounted **<** Object

Texture with optional normal and specular maps for use in 2D rendering.

Description

CanvasTexture is an alternative to ImageTexture for 2D rendering. It allows using normal maps and specular maps in any node that inherits from CanvasItem. **CanvasTexture** also allows overriding the texture's filter and repeat mode independently of the node's properties (or the project settings).

Note: CanvasTexture cannot be used in 3D. It will not display correctly when applied to any VisualInstance3D, such as Sprite3D or Decal. For physically-based materials in 3D, use BaseMaterial3D instead.

Tutorials

- 2D Lights and Shadows

Properties

Texture2D	diffuse_texture
Texture2D	normal_texture

bool	resource_local_to_scene	`false` (overrides Resource)
Color	specular_color	`Color(1, 1, 1, 1)`
float	specular_shininess	`1.0`
Texture2D	specular_texture	
TextureFilter	texture_filter	`0`
TextureRepeat	texture_repeat	`0`

Property Descriptions

Texture2D **diffuse_texture** 🔗

- void **set_diffuse_texture**(value: Texture2D)
- Texture2D **get_diffuse_texture**()

The diffuse (color) texture to use. This is the main texture you want to set in most cases.

Texture2D **normal_texture** 🔗

- void **set_normal_texture**(value: Texture2D)
- Texture2D **get_normal_texture**()

The normal map texture to use. Only has a visible effect if Light2Ds are affecting this **CanvasTexture**.

Note: Godot expects the normal map to use X+, Y+, and Z+ coordinates. See this page [http://wiki.polycount.com/wiki/Normal_Map_Technical_Details#Common_Swizzle_Coordinates] for a comparison of normal map coordinates expected by popular engines.

Color **specular_color** = `Color(1, 1, 1, 1)` 🔗

- void **set_specular_color**(value: Color)
- Color **get_specular_color**()

The multiplier for specular reflection colors. The Light2D's color is also taken into account when determining the reflection color. Only has a visible effect if Light2Ds are affecting this **CanvasTexture**.

---

float **specular_shininess** = `1.0` 🔗

- void **set_specular_shininess**(value: float)
- float **get_specular_shininess**()

The specular exponent for Light2D specular reflections. Higher values result in a more glossy/"wet" look, with reflections becoming more localized and less visible overall. The default value of `1.0` disables specular reflections entirely. Only has a visible effect if Light2Ds are affecting this **CanvasTexture**.

Texture2D **specular_texture** 🔗

- void **set_specular_texture**(value: Texture2D)
- Texture2D **get_specular_texture**()

The specular map to use for Light2D specular reflections. This should be a grayscale or colored texture, with brighter areas resulting in a higher specular_shininess value. Using a colored specular_texture allows controlling specular shininess on a per-channel basis. Only has a visible effect if Light2Ds are affecting this **CanvasTexture**.

TextureFilter **texture_filter** = 0

- void **set_texture_filter**(value: TextureFilter)
- TextureFilter **get_texture_filter**()

The texture filtering mode to use when drawing this **CanvasTexture**.

TextureRepeat **texture_repeat** = 0

- void **set_texture_repeat**(value: TextureRepeat)
- TextureRepeat **get_texture_repeat**()

The texture repeat mode to use when drawing this **CanvasTexture**.

CapsuleMesh

Inherits: PrimitiveMesh **<** Mesh **<** Resource **<** RefCounted **<** Object

Class representing a capsule-shaped PrimitiveMesh.

Description

Class representing a capsule-shaped PrimitiveMesh.

Properties

float	height	`2.0`
int	radial_segments	`64`
float	radius	`0.5`
int	rings	`8`

Property Descriptions

float **height** = `2.0` 🔗

- void **set_height**(value: float)

- float **get_height**()

Total height of the capsule mesh (including the hemispherical ends).

---

int **radial_segments** = `64` 🔗

- void **set_radial_segments**(value: int)
- int **get_radial_segments**()

Number of radial segments on the capsule mesh.

---

float **radius** = `0.5` 🔗

- void **set_radius**(value: float)
- float **get_radius**()

Radius of the capsule mesh.

---

int **rings** = `8` 🔗

- void **set_rings**(value: int)
- int **get_rings**()

Number of rings along the height of the capsule.

CapsuleShape2D

Inherits: Shape2D **<** Resource **<** RefCounted **<** Object

A 2D capsule shape used for physics collision.

Description

A 2D capsule shape, intended for use in physics. Usually used to provide a shape for a CollisionShape2D.

Performance: CapsuleShape2D is fast to check collisions against, but it is slower than RectangleShape2D and CircleShape2D.

Properties

float	height	`30.0`
float	radius	`10.0`

Property Descriptions

float **height** = `30.0`

- void **set_height**(value: float)
- float **get_height**()

The capsule's height.

---

float **radius** = `10.0` 🔗

- void **set_radius**(value: float)
- float **get_radius**()

The capsule's radius.

CapsuleShape3D

Inherits: Shape3D **<** Resource **<** RefCounted **<** Object

A 3D capsule shape used for physics collision.

Description

A 3D capsule shape, intended for use in physics. Usually used to provide a shape for a CollisionShape3D.

Performance: CapsuleShape3D is fast to check collisions against. It is faster than CylinderShape3D, but slower than SphereShape3D and BoxShape3D.

Tutorials

- 3D Physics Tests Demo [https://godotengine.org/asset-library/asset/2747]

Properties

float	height	`2.0`
float	radius	`0.5`

Property Descriptions

float **height** = `2.0` 🔗

- void **set_height**(value: float)
- float **get_height**()

The capsule's height.

---

float **radius** = `0.5` 🔗

- void **set_radius**(value: float)
- float **get_radius**()

The capsule's radius.

CircleShape2D

Inherits: Shape2D **<** Resource **<** RefCounted **<** Object

A 2D circle shape used for physics collision.

Description

A 2D circle shape, intended for use in physics. Usually used to provide a shape for a CollisionShape2D.

Performance: CircleShape2D is fast to check collisions against. It is faster than RectangleShape2D and CapsuleShape2D.

Properties

float	radius	`10.0`

Property Descriptions

float **radius** = `10.0`

- void **set_radius**(value: float)
- float **get_radius**()

The circle's radius.

CodeHighlighter

Inherits: SyntaxHighlighter **<** Resource **<** RefCounted **<** Object

A syntax highlighter intended for code.

Description

By adjusting various properties of this resource, you can change the colors of strings, comments, numbers, and other text patterns inside a TextEdit control.

Properties

Dictionary	color_regions	`{}`
Color	function_color	`Color(0, 0, 0, 1)`
Dictionary	keyword_colors	`{}`
Dictionary	member_keyword_colors	`{}`
Color	member_variable_color	`Color(0, 0, 0, 1)`
Color	number_color	`Color(0, 0, 0, 1)`

Color	symbol_color	`Color(0, 0, 0, 1)`

Methods

void	add_color_region(start_key: String, end_key: String, color: Color, line_only: bool = false)
void	add_keyword_color(keyword: String, color: Color)
void	add_member_keyword_color(member_keyword: String, color: Color)
void	clear_color_regions()
void	clear_keyword_colors()
void	clear_member_keyword_colors()
Color	get_keyword_color(keyword: String) const
Color	get_member_keyword_color(member_keyword: String) const
bool	has_color_region(start_key: String) const

bool	has_keyword_color(keyword: String) const
bool	has_member_keyword_color(member_keyword: String) const
void	remove_color_region(start_key: String)
void	remove_keyword_color(keyword: String)
void	remove_member_keyword_color(member_keyword: String)

Property Descriptions

Dictionary **color_regions** = `{}`

- void **set_color_regions**(value: Dictionary)
- Dictionary **get_color_regions**()

Sets the color regions. All existing regions will be removed. The Dictionary key is the region start and end key, separated by a space. The value is the region color.

Color **function_color** = `Color(0, 0, 0, 1)`

- void **set_function_color**(value: Color)
- Color **get_function_color**()

Sets color for functions. A function is a non-keyword string followed by a '('.

Dictionary **keyword_colors** = `{}`

- void **set_keyword_colors**(value: Dictionary)
- Dictionary **get_keyword_colors**()

Sets the keyword colors. All existing keywords will be removed. The Dictionary key is the keyword. The value is the keyword color.

Dictionary **member_keyword_colors** = `{}`

- void **set_member_keyword_colors**(value: Dictionary)
- Dictionary **get_member_keyword_colors**()

Sets the member keyword colors. All existing member keyword will be removed. The Dictionary key is the member keyword. The value is the member keyword color.

Color **member_variable_color** = `Color(0, 0, 0, 1)`

- void **set_member_variable_color**(value: Color)
- Color **get_member_variable_color**()

Sets color for member variables. A member variable is non-keyword, non-function string proceeded with a '.'.

Color **number_color** = `Color(0, 0, 0, 1)`

- void **set_number_color**(value: Color)
- Color **get_number_color**()

Sets the color for numbers.

Color **symbol_color** = `Color(0, 0, 0, 1)` 🔗

- void **set_symbol_color**(value: Color)
- Color **get_symbol_color**()

Sets the color for symbols.

Method Descriptions

void **add_color_region**(start_key: String, end_key: String, color: Color, line_only: bool = false) 🔗

Adds a color region (such as for comments or strings) from `start_key` to `end_key`. Both keys should be symbols, and `start_key` must not be shared with other delimiters.

If `line_only` is `true` or `end_key` is an empty String, the region does not carry over to the next line.

---

void **add_keyword_color**(keyword: String, color: Color) 🔗

Sets the color for a keyword.

The keyword cannot contain any symbols except '_'.

---

void **add_member_keyword_color**(member_keyword: String, color: Color) 🔗

Sets the color for a member keyword.

The member keyword cannot contain any symbols except '_'.

It will not be highlighted if preceded by a '.'.

void **clear_color_regions**() 🔗

Removes all color regions.

void **clear_keyword_colors**() 🔗

Removes all keywords.

void **clear_member_keyword_colors**() 🔗

Removes all member keywords.

Color **get_keyword_color**(keyword: String) const 🔗

Returns the color for a keyword.

Color **get_member_keyword_color**(member_keyword: String) const 🔗

Returns the color for a member keyword.

bool **has_color_region**(start_key: String) const 🔗

Returns `true` if the start key exists, else `false`.

bool **has_keyword_color**(keyword: String) const 🔗

Returns `true` if the keyword exists, else `false`.

bool **has_member_keyword_color**(member_keyword: String) const

Returns `true` if the member keyword exists, else `false`.

---

void **remove_color_region**(start_key: String)

Removes the color region that uses that start key.

---

void **remove_keyword_color**(keyword: String)

Removes the keyword.

---

void **remove_member_keyword_color**(member_keyword: String)

Removes the member keyword.

Compositor

Experimental: More customization of the rendering pipeline will be added in the future.

Inherits: Resource **<** RefCounted **<** Object

Stores attributes used to customize how a Viewport is rendered.

Description

The compositor resource stores attributes used to customize how a Viewport is rendered.

Properties

Array[CompositorEffect]	compositor_effects	`[]`

Property Descriptions

Array[CompositorEffect] **compositor_effects** = `[]` 🔗

- void **set_compositor_effects**(value: Array[CompositorEffect])
- Array[CompositorEffect] **get_compositor_effects**()

The custom CompositorEffects that are applied during rendering of viewports using this compositor.

CompositorEffect

Experimental: The implementation may change as more of the rendering internals are exposed over time.

Inherits: Resource **<** RefCounted **<** Object

This resource allows for creating a custom rendering effect.

Description

This resource defines a custom rendering effect that can be applied to Viewports through the viewports' Environment. You can implement a callback that is called during rendering at a given stage of the rendering pipeline and allows you to insert additional passes. Note that this callback happens on the rendering thread. CompositorEffect is an abstract base class and must be extended to implement specific rendering logic.

Properties

bool	access_resolved_color
bool	access_resolved_depth
EffectCallbackType	effect_callback_type
bool	enabled

bool	needs_motion_vectors
bool	needs_normal_roughness
bool	needs_separate_specular

Methods

void	_render_callback(effect_callback_type: int, render_data: RenderData) virtual

Enumerations

enum **EffectCallbackType**:

EffectCallbackType **EFFECT_CALLBACK_TYPE_PRE_OPAQUE** = `0`

The callback is called before our opaque rendering pass, but after depth prepass (if applicable).

EffectCallbackType **EFFECT_CALLBACK_TYPE_POST_OPAQUE** = `1`

The callback is called after our opaque rendering pass, but before our sky is rendered.

EffectCallbackType **EFFECT_CALLBACK_TYPE_POST_SKY** = `2`

The callback is called after our sky is rendered, but before our back buffers are created (and if enabled, before subsurface scattering and/or screen space reflections).

EffectCallbackType **EFFECT_CALLBACK_TYPE_PRE_TRANSPARENT** = `3`

The callback is called before our transparent rendering pass, but after our sky is rendered and we've created our back buffers.

EffectCallbackType **EFFECT_CALLBACK_TYPE_POST_TRANSPARENT** = `4`

The callback is called after our transparent rendering pass, but before any build in post effects and output to our render target.

EffectCallbackType **EFFECT_CALLBACK_TYPE_MAX** = `5`

Represents the size of the EffectCallbackType enum.

Property Descriptions

bool **access_resolved_color** 🔗

- void **set_access_resolved_color**(value: bool)
- bool **get_access_resolved_color**()

If `true` and MSAA is enabled, this will trigger a color buffer resolve before the effect is run.

Note: In _render_callback, to access the resolved buffer use:

```
var render_scene_buffers : RenderSceneBuffersRD =
render_data.get_render_scene_buffers()
```

```
var color_buffer =
render_scene_buffers.get_texture("render_buffers", "color")
```

---

bool **access_resolved_depth** 🔗

- void **set_access_resolved_depth**(value: bool)
- bool **get_access_resolved_depth**()

If `true` and MSAA is enabled, this will trigger a depth buffer resolve before the effect is run.

Note: In _render_callback, to access the resolved buffer use:

```
var render_scene_buffers : RenderSceneBuffersRD =
render_data.get_render_scene_buffers()
var depth_buffer =
render_scene_buffers.get_texture("render_buffers", "depth")
```

EffectCallbackType **effect_callback_type** 🔗

- void **set_effect_callback_type**(value: EffectCallbackType)
- EffectCallbackType **get_effect_callback_type**()

The type of effect that is implemented, determines at what stage of rendering the callback is called.

---

bool **enabled** 🔗

- void **set_enabled**(value: bool)
- bool **get_enabled**()

If `true` this rendering effect is applied to any viewport it is added to.

---

bool **needs_motion_vectors** 🔗

- void **set_needs_motion_vectors**(value: bool)
- bool **get_needs_motion_vectors**()

If `true` this triggers motion vectors being calculated during the opaque render state.

Note: In _render_callback, to access the motion vector buffer use:

```
var render_scene_buffers : RenderSceneBuffersRD =
render_data.get_render_scene_buffers()
var motion_buffer = render_scene_buffers.get_velocity_texture()
```

---

bool **needs_normal_roughness** 🔗

- void **set_needs_normal_roughness**(value: bool)
- bool **get_needs_normal_roughness**()

If `true` this triggers normal and roughness data to be output during our depth pre-pass, only applicable for the Forward+ renderer.

Note: In _render_callback, to access the roughness buffer use:

```
var render_scene_buffers : RenderSceneBuffersRD =
render_data.get_render_scene_buffers()
var roughness_buffer =
render_scene_buffers.get_texture("forward_clustered",
"normal_roughness")
```

---

bool **needs_separate_specular** 🔗

- void **set_needs_separate_specular**(value: bool)
- bool **get_needs_separate_specular**()

If `true` this triggers specular data being rendered to a separate buffer and combined after effects have been applied, only applicable for the Forward+ renderer.

Method Descriptions

void **_render_callback**(effect_callback_type: int, render_data: RenderData) virtual 🔗

Implement this function with your custom rendering code. `effect_callback_type` should always match the effect callback type you've specified in effect_callback_type. `render_data` provides access to the rendering state, it is only valid during rendering and should not be stored.

CompressedCubemap

Inherits: CompressedTextureLayered **<** TextureLayered **<** Texture **<** Resource **<** RefCounted **<** Object

An optionally compressed Cubemap.

Description

A cubemap that is loaded from a `.ccube` file. This file format is internal to Godot; it is created by importing other image formats with the import system. **CompressedCubemap** can use one of 4 compression methods:

- Lossless (WebP or PNG, uncompressed on the GPU)
- Lossy (WebP, uncompressed on the GPU)
- VRAM Compressed (compressed on the GPU)
- VRAM Uncompressed (uncompressed on the GPU)
- Basis Universal (compressed on the GPU. Lower file sizes than VRAM Compressed, but slower to compress and lower quality than VRAM Compressed)

Only **VRAM Compressed** actually reduces the memory usage on the GPU. The **Lossless** and **Lossy** compression methods will reduce the required storage on disk, but they will not reduce memory usage on the GPU as the texture is sent to the GPU uncompressed.

Using **VRAM Compressed** also improves loading times, as VRAM-compressed textures are faster to load compared to textures using lossless or lossy compression. VRAM compression can exhibit noticeable artifacts and is intended to be used for 3D rendering, not 2D.

See Cubemap for a general description of cubemaps.

CompressedCubemapArray

Inherits: CompressedTextureLayered **<** TextureLayered **<** Texture **<** Resource **<** RefCounted **<** Object

An optionally compressed CubemapArray.

Description

A cubemap array that is loaded from a `.ccubearray` file. This file format is internal to Godot; it is created by importing other image formats with the import system. **CompressedCubemapArray** can use one of 4 compression methods:

- Lossless (WebP or PNG, uncompressed on the GPU)
- Lossy (WebP, uncompressed on the GPU)
- VRAM Compressed (compressed on the GPU)
- VRAM Uncompressed (uncompressed on the GPU)
- Basis Universal (compressed on the GPU. Lower file sizes than VRAM Compressed, but slower to compress and lower quality than VRAM Compressed)

Only **VRAM Compressed** actually reduces the memory usage on the GPU. The **Lossless** and **Lossy** compression methods will reduce the required storage on disk, but they will not reduce memory usage on the GPU as the texture is sent to the GPU uncompressed.

Using **VRAM Compressed** also improves loading times, as VRAM-compressed textures are faster to load compared to textures using lossless or lossy compression. VRAM compression can exhibit noticeable artifacts and is intended to be used for 3D rendering, not 2D.

See CubemapArray for a general description of cubemap arrays.

CompressedTexture2D

Inherits: Texture2D **<** Texture **<** Resource **<** RefCounted **<** Object

Texture with 2 dimensions, optionally compressed.

Description

A texture that is loaded from a `.ctex` file. This file format is internal to Godot; it is created by importing other image formats with the import system. **CompressedTexture2D** can use one of 4 compression methods (including a lack of any compression):

- Lossless (WebP or PNG, uncompressed on the GPU)
- Lossy (WebP, uncompressed on the GPU)
- VRAM Compressed (compressed on the GPU)
- VRAM Uncompressed (uncompressed on the GPU)
- Basis Universal (compressed on the GPU. Lower file sizes than VRAM Compressed, but slower to compress and lower quality than VRAM Compressed)

Only **VRAM Compressed** actually reduces the memory usage on the GPU. The **Lossless** and **Lossy** compression methods will reduce the required storage on disk, but they will not reduce memory usage on the GPU as the texture is sent to the GPU uncompressed.

Using **VRAM Compressed** also improves loading times, as VRAM-compressed textures are faster to load compared to textures using lossless or lossy compression. VRAM compression can exhibit noticeable artifacts and is intended to be used for 3D rendering, not 2D.

Properties

String	load_path	`""`
bool	resource_local_to_scene	`false` (overrides Resource)

Methods

Error	load(path: String)

Property Descriptions

String **load_path** = `""` 🔗

- Error **load**(path: String)
- String **get_load_path**()

The **CompressedTexture2D**'s file path to a `.ctex` file.

Method Descriptions

Error **load**(path: String) 🔗

Loads the texture from the specified `path`.

CompressedTexture2DArray

Inherits: CompressedTextureLayered **<** TextureLayered **<** Texture **<** Resource **<** RefCounted **<** Object

Array of 2-dimensional textures, optionally compressed.

Description

A texture array that is loaded from a `.ctexarray` file. This file format is internal to Godot; it is created by importing other image formats with the import system. **CompressedTexture2DArray** can use one of 4 compression methods:

- Lossless (WebP or PNG, uncompressed on the GPU)
- Lossy (WebP, uncompressed on the GPU)
- VRAM Compressed (compressed on the GPU)
- VRAM Uncompressed (uncompressed on the GPU)
- Basis Universal (compressed on the GPU. Lower file sizes than VRAM Compressed, but slower to compress and lower quality than VRAM Compressed)

Only **VRAM Compressed** actually reduces the memory usage on the GPU. The **Lossless** and **Lossy** compression methods will reduce the required storage on disk, but they will not reduce memory usage on the GPU as the texture is sent to the GPU uncompressed.

Using **VRAM Compressed** also improves loading times, as VRAM-compressed textures are faster to load compared to textures using lossless or lossy compression. VRAM compression can exhibit noticeable artifacts and is intended to be used for 3D rendering, not 2D.

See Texture2DArray for a general description of texture arrays.

CompressedTexture3D

Inherits: Texture3D **<** Texture **<** Resource **<** RefCounted **<** Object

Texture with 3 dimensions, optionally compressed.

Description

CompressedTexture3D is the VRAM-compressed counterpart of ImageTexture3D. The file extension for **CompressedTexture3D** files is `.ctex3d`. This file format is internal to Godot; it is created by importing other image formats with the import system.

CompressedTexture3D uses VRAM compression, which allows to reduce memory usage on the GPU when rendering the texture. This also improves loading times, as VRAM-compressed textures are faster to load compared to textures using lossless compression. VRAM compression can exhibit noticeable artifacts and is intended to be used for 3D rendering, not 2D.

See Texture3D for a general description of 3D textures.

Properties

String	load_path	`""`

Methods

Error load(path: String)

Property Descriptions

String **load_path** = `""` 🔗

- Error **load**(path: String)
- String **get_load_path**()

The **CompressedTexture3D**'s file path to a `.ctex3d` file.

Method Descriptions

Error **load**(path: String) 🔗

Loads the texture from the specified `path`.

CompressedTextureLayered

Inherits: TextureLayered **<** Texture **<** Resource **<** RefCounted **<** Object

Inherited By: CompressedCubemap, CompressedCubemapArray, CompressedTexture2DArray

Base class for texture arrays that can optionally be compressed.

Description

Base class for CompressedTexture2DArray and CompressedTexture3D. Cannot be used directly, but contains all the functions necessary for accessing the derived resource types. See also TextureLayered.

Properties

String	load_path	""

Methods

Error	load(path: String)

Property Descriptions

String **load_path** = `""`

- Error **load**(path: String)
- String **get_load_path**()

The path the texture should be loaded from.

Method Descriptions

Error **load**(path: String)

Loads the texture at `path`.

ConcavePolygonShape2D

Inherits: Shape2D **<** Resource **<** RefCounted **<** Object

A 2D polyline shape used for physics collision.

Description

A 2D polyline shape, intended for use in physics. Used internally in CollisionPolygon2D when it's in CollisionPolygon2D.BUILD_SEGMENTS mode.

Being just a collection of interconnected line segments, **ConcavePolygonShape2D** is the most freely configurable single 2D shape. It can be used to form polygons of any nature, or even shapes that don't enclose an area. However, **ConcavePolygonShape2D** is *hollow* even if the interconnected line segments do enclose an area, which often makes it unsuitable for physics or detection.

Note: When used for collision, **ConcavePolygonShape2D** is intended to work with static CollisionShape2D nodes like StaticBody2D and will likely not behave well for CharacterBody2Ds or RigidBody2Ds in a mode other than Static.

Warning: Physics bodies that are small have a chance to clip through this shape when moving fast. This happens because on one frame, the physics body may be on the "outside" of the shape, and on the next frame it may be "inside" it. **ConcavePolygonShape2D** is hollow, so it won't detect a collision.

Performance: Due to its complexity, **ConcavePolygonShape2D** is the slowest 2D collision shape to check collisions against. Its use should generally be limited to level geometry. If the polyline is closed, CollisionPolygon2D's CollisionPolygon2D.BUILD_SOLIDS mode can be used, which decomposes the polygon into convex ones; see ConvexPolygonShape2D's documentation for instructions.

Properties

PackedVector2Array	segments	`PackedVector2Array()`

Property Descriptions

PackedVector2Array **segments** = `PackedVector2Array()` 🔗

- void **set_segments**(value: PackedVector2Array)
- PackedVector2Array **get_segments**()

The array of points that make up the **ConcavePolygonShape2D**'s line segments. The array (of length divisible by two) is naturally divided into pairs (one pair for each segment); each pair consists of the starting point of a segment and the endpoint of a segment.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector2Array for more details.

ConcavePolygonShape3D

Inherits: Shape3D **<** Resource **<** RefCounted **<** Object

A 3D trimesh shape used for physics collision.

Description

A 3D trimesh shape, intended for use in physics. Usually used to provide a shape for a CollisionShape3D.

Being just a collection of interconnected triangles, **ConcavePolygonShape3D** is the most freely configurable single 3D shape. It can be used to form polyhedra of any nature, or even shapes that don't enclose a volume. However, **ConcavePolygonShape3D** is *hollow* even if the interconnected triangles do enclose a volume, which often makes it unsuitable for physics or detection.

Note: When used for collision, **ConcavePolygonShape3D** is intended to work with static CollisionShape3D nodes like StaticBody3D and will likely not behave well for CharacterBody3Ds or RigidBody3Ds in a mode other than Static.

Warning: Physics bodies that are small have a chance to clip through this shape when moving fast. This happens because on one frame, the physics body may be on the "outside" of the shape, and on the next frame it may be "inside" it. **ConcavePolygonShape3D** is hollow, so it won't detect a collision.

Performance: Due to its complexity, **ConcavePolygonShape3D** is the slowest 3D collision shape to check collisions against. Its use should generally be limited to level geometry. For convex geometry, ConvexPolygonShape3D should be used. For dynamic physics bodies that need concave collision, several ConvexPolygonShape3Ds can be used to represent its collision by using convex decomposition; see ConvexPolygonShape3D's documentation for instructions.

Tutorials

- 3D Physics Tests Demo [https://godotengine.org/asset-library/asset/2747]

Properties

bool	backface_collision	`false`

Methods

PackedVector3Array	get_faces() const
void	set_faces(faces: PackedVector3Array)

Property Descriptions

bool **backface_collision** = `false` 🔗

- void **set_backface_collision_enabled**(value: bool)
- bool **is_backface_collision_enabled**()

If set to `true`, collisions occur on both sides of the concave shape faces. Otherwise they occur only along the face normals.

Method Descriptions

PackedVector3Array **get_faces**() const

Returns the faces of the trimesh shape as an array of vertices. The array (of length divisible by three) is naturally divided into triples; each triple of vertices defines a triangle.

void **set_faces**(faces: PackedVector3Array)

Sets the faces of the trimesh shape from an array of vertices. The `faces` array should be composed of triples such that each triple of vertices defines a triangle.

ConvexPolygonShape2D

Inherits: Shape2D **<** Resource **<** RefCounted **<** Object

A 2D convex polygon shape used for physics collision.

Description

A 2D convex polygon shape, intended for use in physics. Used internally in CollisionPolygon2D when it's in CollisionPolygon2D.BUILD_SOLIDS mode.

ConvexPolygonShape2D is *solid*, which means it detects collisions from objects that are fully inside it, unlike ConcavePolygonShape2D which is hollow. This makes it more suitable for both detection and physics.

Convex decomposition: A concave polygon can be split up into several convex polygons. This allows dynamic physics bodies to have complex concave collisions (at a performance cost) and can be achieved by using several **ConvexPolygonShape2D** nodes or by using the CollisionPolygon2D node in CollisionPolygon2D.BUILD_SOLIDS mode. To generate a collision polygon from a sprite, select the Sprite2D node, go to the **Sprite2D** menu that appears above the viewport, and choose **Create Polygon2D Sibling**.

Performance: ConvexPolygonShape2D is faster to check collisions against compared to ConcavePolygonShape2D, but it is slower than primitive collision shapes such as CircleShape2D and RectangleShape2D. Its use should generally be limited to medium-sized objects that cannot have their collision accurately represented by primitive shapes.

Properties

PackedVector2Array	points	`PackedVector2Array()`

Methods

void	set_point_cloud(point_cloud: PackedVector2Array)

Property Descriptions

PackedVector2Array **points** = `PackedVector2Array()` 🔗

- void **set_points**(value: PackedVector2Array)
- PackedVector2Array **get_points**()

The polygon's list of vertices that form a convex hull. Can be in either clockwise or counterclockwise order.

Warning: Only set this property to a list of points that actually form a convex hull. Use set_point_cloud to generate the convex hull of an arbitrary set of points.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector2Array for more details.

Method Descriptions

void **set_point_cloud**(point_cloud: PackedVector2Array) 🔗

Based on the set of points provided, this assigns the points property using the convex hull algorithm, removing all unneeded points. See Geometry2D.convex_hull for details.

ConvexPolygonShape3D

Inherits: Shape3D **<** Resource **<** RefCounted **<** Object

A 3D convex polyhedron shape used for physics collision.

Description

A 3D convex polyhedron shape, intended for use in physics. Usually used to provide a shape for a CollisionShape3D.

ConvexPolygonShape3D is *solid*, which means it detects collisions from objects that are fully inside it, unlike ConcavePolygonShape3D which is hollow. This makes it more suitable for both detection and physics.

Convex decomposition: A concave polyhedron can be split up into several convex polyhedra. This allows dynamic physics bodies to have complex concave collisions (at a performance cost) and can be achieved by using several **ConvexPolygonShape3D** nodes. To generate a convex decomposition from a mesh, select the MeshInstance3D node, go to the **Mesh** menu that appears above the viewport, and choose **Create Multiple Convex Collision Siblings**. Alternatively, MeshInstance3D.create_multiple_convex_collisions can be called in a script to perform this decomposition at run-time.

Performance: ConvexPolygonShape3D is faster to check collisions against compared to ConcavePolygonShape3D, but it is slower than primitive collision shapes such as SphereShape3D and BoxShape3D. Its use should generally be limited to medium-sized objects that cannot have their collision accurately represented by primitive shapes.

Tutorials

- 3D Physics Tests Demo [https://godotengine.org/asset-library/asset/2747]

Properties

PackedVector3Array	points	`PackedVector3Array()`

Property Descriptions

PackedVector3Array **points** = `PackedVector3Array()`

- void **set_points**(value: PackedVector3Array)
- PackedVector3Array **get_points**()

The list of 3D points forming the convex polygon shape.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector3Array for more details.

CryptoKey

Inherits: Resource **<** RefCounted **<** Object

A cryptographic key (RSA or elliptic-curve).

Description

The CryptoKey class represents a cryptographic key. Keys can be loaded and saved like any other Resource.

They can be used to generate a self-signed X509Certificate via Crypto.generate_self_signed_certificate and as private key in StreamPeerTLS.accept_stream along with the appropriate certificate.

Tutorials

- SSL certificates

Methods

bool	is_public_only() const
Error	load(path: String, public_only: bool = false)
Error	load_from_string(string_key: String, public_only: bool = false)

Error	save(path: String, public_only: bool = false)
String	save_to_string(public_only: bool = false)

Method Descriptions

bool **is_public_only**() const 🔗

Returns `true` if this CryptoKey only has the public part, and not the private one.

Error **load**(path: String, public_only: bool = false) 🔗

Loads a key from `path`. If `public_only` is `true`, only the public key will be loaded.

Note: `path` should be a "*.pub" file if `public_only` is `true`, a "*.key" file otherwise.

Error **load_from_string**(string_key: String, public_only: bool = false) 🔗

Loads a key from the given `string_key`. If `public_only` is `true`, only the public key will be loaded.

Error **save**(path: String, public_only: bool = false) 🔗

Saves a key to the given `path`. If `public_only` is `true`, only the public key will be saved.

Note: `path` should be a "*.pub" file if `public_only` is `true`, a "*.key" file otherwise.

String **save_to_string**(public_only: bool = false) 🔗

Returns a string containing the key in PEM format. If `public_only` is `true`, only the public key will be included.

CSharpScript

Inherits: Script **<** Resource **<** RefCounted **<** Object

A script implemented in the C# programming language, saved with the `.cs` extension (Mono-enabled builds only).

Description

This class represents a C# script. It is the C# equivalent of the GDScript class and is only available in Mono-enabled Godot builds.

Tutorials

- C# documentation index

Methods

Variant	new(...) vararg

Method Descriptions

Variant **new**(...) vararg 🔗

Returns a new instance of the script.

Cubemap

Inherits: ImageTextureLayered **<** TextureLayered **<** Texture **<** Resource **<** RefCounted **<** Object

Six square textures representing the faces of a cube. Commonly used as a skybox.

Description

A cubemap is made of 6 textures organized in layers. They are typically used for faking reflections in 3D rendering (see ReflectionProbe). It can be used to make an object look as if it's reflecting its surroundings. This usually delivers much better performance than other reflection methods.

This resource is typically used as a uniform in custom shaders. Few core Godot methods make use of **Cubemap** resources.

To create such a texture file yourself, reimport your image files using the Godot Editor import presets.

Note: Godot doesn't support using cubemaps in a PanoramaSkyMaterial. You can use this tool [https://danilw.github.io/GLSL-howto/cubemap_to_panorama_js/cubemap_to_panorama.html] to convert a cubemap to an equirectangular sky map.

Methods

Resource	create_placeholder() const

Method Descriptions

Resource **create_placeholder**() const 🔗

Creates a placeholder version of this resource (PlaceholderCubemap).

CubemapArray

Inherits: ImageTextureLayered **<** TextureLayered **<** Texture **<** Resource **<** RefCounted **<** Object

An array of Cubemaps, stored together and with a single reference.

Description

CubemapArrays are made of an array of Cubemaps. Like Cubemaps, they are made of multiple textures, the amount of which must be divisible by 6 (one for each face of the cube).

The primary benefit of **CubemapArrays** is that they can be accessed in shader code using a single texture reference. In other words, you can pass multiple Cubemaps into a shader using a single **CubemapArray**. Cubemaps are allocated in adjacent cache regions on the GPU, which makes **CubemapArrays** the most efficient way to store multiple Cubemaps.

Note: Godot uses **CubemapArrays** internally for many effects, including the Sky if you set ProjectSettings.rendering/reflections/sky_reflections/texture_array_reflections to `true`. To create such a texture file yourself, reimport your image files using the import presets of the File System dock.

Note: CubemapArray is not supported in the OpenGL 3 rendering backend.

Methods

Resource	create_placeholder() const

Method Descriptions

Resource **create_placeholder**() const 🔗

Creates a placeholder version of this resource (PlaceholderCubemapArray).

Curve

Inherits: Resource **<** RefCounted **<** Object

A mathematical curve.

Description

This resource describes a mathematical curve by defining a set of points and tangents at each point. By default, it ranges between `0` and `1` on the Y axis and positions points relative to the `0.5` Y position.

See also Gradient which is designed for color interpolation. See also Curve2D and Curve3D.

Properties

int	bake_resolution	`100`
float	max_value	`1.0`
float	min_value	`0.0`
int	point_count	`0`

Methods

int	add_point(position: Vector2, left_tangent: float = 0, right_tangent: float = 0, left_mode: TangentMode = 0, right_mode: TangentMode = 0)
void	bake()
void	clean_dupes()
void	clear_points()
TangentMode	get_point_left_mode(index: int) const
float	get_point_left_tangent(index: int) const
Vector2	get_point_position(index: int) const
TangentMode	get_point_right_mode(index: int) const
float	get_point_right_tangent(index: int) const
void	remove_point(index: int)
float	sample(offset: float) const

float	sample_baked(offset: float) const
void	set_point_left_mode(index: int, mode: TangentMode)
void	set_point_left_tangent(index: int, tangent: float)
int	set_point_offset(index: int, offset: float)
void	set_point_right_mode(index: int, mode: TangentMode)
void	set_point_right_tangent(index: int, tangent: float)
void	set_point_value(index: int, y: float)

Signals

range_changed() 🔗

Emitted when max_value or min_value is changed.

Enumerations

enum **TangentMode**: 🔗

TangentMode **TANGENT_FREE** = `0`

The tangent on this side of the point is user-defined.

TangentMode **TANGENT_LINEAR** = `1`

The curve calculates the tangent on this side of the point as the slope halfway towards the adjacent point.

TangentMode **TANGENT_MODE_COUNT** = `2`

The total number of available tangent modes.

Property Descriptions

int **bake_resolution** = `100`

- void **set_bake_resolution**(value: int)
- int **get_bake_resolution**()

The number of points to include in the baked (i.e. cached) curve data.

---

float **max_value** = `1.0`

- void **set_max_value**(value: float)
- float **get_max_value**()

The maximum value the curve can reach.

---

float **min_value** = `0.0`

- void **set_min_value**(value: float)

- float **get_min_value**()

The minimum value the curve can reach.

int **point_count** = `0`

- void **set_point_count**(value: int)
- int **get_point_count**()

The number of points describing the curve.

Method Descriptions

int **add_point**(position: Vector2, left_tangent: float = 0, right_tangent: float = 0, left_mode: TangentMode = 0, right_mode: TangentMode = 0)

Adds a point to the curve. For each side, if the `*_mode` is TANGENT_LINEAR, the `*_tangent` angle (in degrees) uses the slope of the curve halfway to the adjacent point. Allows custom assignments to the `*_tangent` angle if `*_mode` is set to TANGENT_FREE.

void **bake**()

Recomputes the baked cache of points for the curve.

void **clean_dupes**()

Removes duplicate points, i.e. points that are less than 0.00001 units (engine epsilon value) away from their neighbor on the curve.

void **clear_points**() 🔗

Removes all points from the curve.

TangentMode **get_point_left_mode**(index: int) const 🔗

Returns the left TangentMode for the point at `index`.

float **get_point_left_tangent**(index: int) const 🔗

Returns the left tangent angle (in degrees) for the point at `index`.

Vector2 **get_point_position**(index: int) const 🔗

Returns the curve coordinates for the point at `index`.

TangentMode **get_point_right_mode**(index: int) const 🔗

Returns the right TangentMode for the point at `index`.

float **get_point_right_tangent**(index: int) const 🔗

Returns the right tangent angle (in degrees) for the point at `index`.

void **remove_point**(index: int) 🔗

Removes the point at `index` from the curve.

float **sample**(offset: float) const 🔗

Returns the Y value for the point that would exist at the X position `offset` along the curve.

---

float **sample_baked**(offset: float) const

Returns the Y value for the point that would exist at the X position `offset` along the curve using the baked cache. Bakes the curve's points if not already baked.

---

void **set_point_left_mode**(index: int, mode: TangentMode)

Sets the left TangentMode for the point at `index` to `mode`.

---

void **set_point_left_tangent**(index: int, tangent: float)

Sets the left tangent angle for the point at `index` to `tangent`.

---

int **set_point_offset**(index: int, offset: float)

Sets the offset from `0.5`.

---

void **set_point_right_mode**(index: int, mode: TangentMode)

Sets the right TangentMode for the point at `index` to `mode`.

---

void **set_point_right_tangent**(index: int, tangent: float)

Sets the right tangent angle for the point at `index` to `tangent`.

---

void **set_point_value**(index: int, y: float)

Assigns the vertical position `y` to the point at `index`.

Curve2D

Inherits: Resource **<** RefCounted **<** Object

Describes a Bézier curve in 2D space.

Description

This class describes a Bézier curve in 2D space. It is mainly used to give a shape to a Path2D, but can be manually sampled for other purposes.

It keeps a cache of precalculated points along the curve, to speed up further calculations.

Properties

float	bake_interval	`5.0`
int	point_count	`0`

Methods

void	add_point(position: Vector2, in: Vector2 = Vector2(0, 0), out: Vector2 = Vector2(0, 0), index: int = -1)

void	clear_points()
float	get_baked_length() const
PackedVector2Array	get_baked_points() const
float	get_closest_offset(to_point: Vector2) const
Vector2	get_closest_point(to_point: Vector2) const
Vector2	get_point_in(idx: int) const
Vector2	get_point_out(idx: int) const
Vector2	get_point_position(idx: int) const
void	remove_point(idx: int)
Vector2	sample(idx: int, t: float) const
Vector2	sample_baked(offset: float = 0.0, cubic: bool = false) const
Transform2D	sample_baked_with_rotation(offset: float = 0.0, cubic: bool = false) const

Vector2	samplef(fofs: float) const
void	set_point_in(idx: int, position: Vector2)
void	set_point_out(idx: int, position: Vector2)
void	set_point_position(idx: int, position: Vector2)
PackedVector2Array	tessellate(max_stages: int = 5, tolerance_degrees: float = 4) const
PackedVector2Array	tessellate_even_length(max_stages: int = 5, tolerance_length: float = 20.0) const

Property Descriptions

float **bake_interval** = `5.0` 🔗

- void **set_bake_interval**(value: float)
- float **get_bake_interval**()

The distance in pixels between two adjacent cached points. Changing it forces the cache to be recomputed the next time the get_baked_points or get_baked_length function is called. The smaller the distance, the more points in the cache and the more memory it will consume, so use with care.

int **point_count** = `0` 🔗

- void **set_point_count**(value: int)
- int **get_point_count**()

The number of points describing the curve.

Method Descriptions

void **add_point**(position: Vector2, in: Vector2 = Vector2(0, 0), out: Vector2 = Vector2(0, 0), index: int = -1) 🔗

Adds a point with the specified `position` relative to the curve's own position, with control points `in` and `out`. Appends the new point at the end of the point list.

If `index` is given, the new point is inserted before the existing point identified by index `index`. Every existing point starting from `index` is shifted further down the list of points. The index must be greater than or equal to `0` and must not exceed the number of existing points in the line. See point_count.

---

void **clear_points**() 🔗

Removes all points from the curve.

---

float **get_baked_length**() const 🔗

Returns the total length of the curve, based on the cached points. Given enough density (see bake_interval), it should be approximate enough.

PackedVector2Array **get_baked_points**() const

Returns the cache of points as a PackedVector2Array.

---

float **get_closest_offset**(to_point: Vector2) const

Returns the closest offset to `to_point`. This offset is meant to be used in sample_baked.

`to_point` must be in this curve's local space.

Vector2 **get_closest_point**(to_point: Vector2) const

Returns the closest point on baked segments (in curve's local space) to `to_point`.

`to_point` must be in this curve's local space.

Vector2 **get_point_in**(idx: int) const

Returns the position of the control point leading to the vertex `idx`. The returned position is relative to the vertex `idx`. If the index is out of bounds, the function sends an error to the console, and returns `(0, 0)`.

Vector2 **get_point_out**(idx: int) const

Returns the position of the control point leading out of the vertex `idx`. The returned position is relative to the vertex `idx`. If the index is out of bounds, the function sends an error to the console, and returns `(0, 0)`.

Vector2 **get_point_position**(idx: int) const 🔗

Returns the position of the vertex `idx`. If the index is out of bounds, the function sends an error to the console, and returns `(0, 0)`.

---

void **remove_point**(idx: int) 🔗

Deletes the point `idx` from the curve. Sends an error to the console if `idx` is out of bounds.

Vector2 **sample**(idx: int, t: float) const 🔗

Returns the position between the vertex `idx` and the vertex `idx + 1`, where `t` controls if the point is the first vertex (`t = 0.0`), the last vertex (`t = 1.0`), or in between. Values of `t` outside the range (`0.0 <= t <= 1.0`) give strange, but predictable results.

If `idx` is out of bounds it is truncated to the first or last vertex, and `t` is ignored. If the curve has no points, the function sends an error to the console, and returns `(0, 0)`.

Vector2 **sample_baked**(offset: float = 0.0, cubic: bool = false) const 🔗

Returns a point within the curve at position `offset`, where `offset` is measured as a pixel distance along the curve.

To do that, it finds the two cached points where the `offset` lies between, then interpolates the values. This interpolation is cubic if `cubic` is set to `true`, or linear if set to `false`.

Cubic interpolation tends to follow the curves better, but linear is faster (and often, precise enough).

Transform2D **sample_baked_with_rotation**(offset: float = 0.0, cubic: bool = false) const

Similar to sample_baked, but returns Transform2D that includes a rotation along the curve, with Transform2D.origin as the point position and the Transform2D.x vector pointing in the direction of the path at that point. Returns an empty transform if the length of the curve is `0`.

```
var baked = curve.sample_baked_with_rotation(offset)
# The returned Transform2D can be set directly.
transform = baked
# You can also read the origin and rotation separately from the
returned Transform2D.
position = baked.get_origin()
rotation = baked.get_rotation()
```

Vector2 **samplef**(fofs: float) const

Returns the position at the vertex `fofs`. It calls sample using the integer part of `fofs` as `idx`, and its fractional part as `t`.

---

void **set_point_in**(idx: int, position: Vector2)

Sets the position of the control point leading to the vertex `idx`. If the index is out of bounds, the function sends an error to the console. The position is relative to the vertex.

---

void **set_point_out**(idx: int, position: Vector2)

Sets the position of the control point leading out of the vertex `idx`. If the index is out of bounds, the function sends an error to the console. The position is relative to the vertex.

---

void **set_point_position**(idx: int, position: Vector2)

Sets the position for the vertex `idx`. If the index is out of bounds, the function sends an error to the console.

PackedVector2Array **tessellate**(max_stages: int = 5, tolerance_degrees: float = 4) const

Returns a list of points along the curve, with a curvature controlled point density. That is, the curvier parts will have more points than the straighter parts.

This approximation makes straight segments between each point, then subdivides those segments until the resulting shape is similar enough.

`max_stages` controls how many subdivisions a curve segment may face before it is considered approximate enough. Each subdivision splits the segment in half, so the default 5 stages may mean up to 32 subdivisions per curve segment. Increase with care!

`tolerance_degrees` controls how many degrees the midpoint of a segment may deviate from the real curve, before the segment has to be subdivided.

PackedVector2Array **tessellate_even_length**(max_stages: int = 5, tolerance_length: float = 20.0) const

Returns a list of points along the curve, with almost uniform density. `max_stages` controls how many subdivisions a curve segment may face before it is considered approximate enough. Each subdivision splits the segment in half, so the default 5 stages may mean up to 32 subdivisions per curve segment. Increase with care!

`tolerance_length` controls the maximal distance between two neighboring points, before the segment has to be subdivided.

Curve3D

Inherits: Resource **<** RefCounted **<** Object

Describes a Bézier curve in 3D space.

Description

This class describes a Bézier curve in 3D space. It is mainly used to give a shape to a Path3D, but can be manually sampled for other purposes.

It keeps a cache of precalculated points along the curve, to speed up further calculations.

Properties

float	bake_interval	`0.2`
int	point_count	`0`
bool	up_vector_enabled	`true`

Methods

void	add_point(position: Vector3, in: Vector3 = Vector3(0, 0, 0), out: Vector3 = Vector3(0, 0, 0), index: int = -1)
void	clear_points()
float	get_baked_length() const
PackedVector3Array	get_baked_points() const
PackedFloat32Array	get_baked_tilts() const
PackedVector3Array	get_baked_up_vectors() const
float	get_closest_offset(to_point: Vector3) const
Vector3	get_closest_point(to_point: Vector3) const
Vector3	get_point_in(idx: int) const
Vector3	get_point_out(idx: int) const
Vector3	get_point_position(idx: int) const
float	get_point_tilt(idx: int) const

void	remove_point(idx: int)
Vector3	sample(idx: int, t: float) const
Vector3	sample_baked(offset: float = 0.0, cubic: bool = false) const
Vector3	sample_baked_up_vector(offset: float, apply_tilt: bool = false) const
Transform3D	sample_baked_with_rotation(offset: float = 0.0, cubic: bool = false, apply_tilt: bool = false) const
Vector3	samplef(fofs: float) const
void	set_point_in(idx: int, position: Vector3)
void	set_point_out(idx: int, position: Vector3)
void	set_point_position(idx: int, position: Vector3)
void	set_point_tilt(idx: int, tilt: float)
PackedVector3Array	tessellate(max_stages: int = 5, tolerance_degrees: float = 4) const

PackedVector3Array	tessellate_even_length(max_stages: int = 5, tolerance_length: float = 0.2) const

Property Descriptions

float **bake_interval** = `0.2`

- void **set_bake_interval**(value: float)
- float **get_bake_interval**()

The distance in meters between two adjacent cached points. Changing it forces the cache to be recomputed the next time the get_baked_points or get_baked_length function is called. The smaller the distance, the more points in the cache and the more memory it will consume, so use with care.

---

int **point_count** = `0`

- void **set_point_count**(value: int)
- int **get_point_count**()

The number of points describing the curve.

---

bool **up_vector_enabled** = `true`

- void **set_up_vector_enabled**(value: bool)
- bool **is_up_vector_enabled**()

If `true`, the curve will bake up vectors used for orientation. This is used when PathFollow3D.rotation_mode is set to

PathFollow3D.ROTATION_ORIENTED. Changing it forces the cache to be recomputed.

Method Descriptions

void **add_point**(position: Vector3, in: Vector3 = Vector3(0, 0, 0), out: Vector3 = Vector3(0, 0, 0), index: int = -1)

Adds a point with the specified `position` relative to the curve's own position, with control points `in` and `out`. Appends the new point at the end of the point list.

If `index` is given, the new point is inserted before the existing point identified by index `index`. Every existing point starting from `index` is shifted further down the list of points. The index must be greater than or equal to `0` and must not exceed the number of existing points in the line. See point_count.

---

void **clear_points**()

Removes all points from the curve.

---

float **get_baked_length**() const

Returns the total length of the curve, based on the cached points. Given enough density (see bake_interval), it should be approximate enough.

PackedVector3Array **get_baked_points**() const

Returns the cache of points as a PackedVector3Array.

PackedFloat32Array **get_baked_tilts**() const 🔗

Returns the cache of tilts as a PackedFloat32Array.

PackedVector3Array **get_baked_up_vectors**() const 🔗

Returns the cache of up vectors as a PackedVector3Array.

If up_vector_enabled is `false`, the cache will be empty.

---

float **get_closest_offset**(to_point: Vector3) const 🔗

Returns the closest offset to `to_point`. This offset is meant to be used in sample_baked or sample_baked_up_vector.

`to_point` must be in this curve's local space.

Vector3 **get_closest_point**(to_point: Vector3) const 🔗

Returns the closest point on baked segments (in curve's local space) to `to_point`.

`to_point` must be in this curve's local space.

Vector3 **get_point_in**(idx: int) const 🔗

Returns the position of the control point leading to the vertex `idx`. The returned position is relative to the vertex `idx`. If the index is out of bounds, the function sends an error to the console, and returns `(0, 0, 0)`.

Vector3 **get_point_out**(idx: int) const 🔗

Returns the position of the control point leading out of the vertex `idx`. The returned position is relative to the vertex `idx`. If the index is out of bounds, the function sends an error to the console, and returns `(0, 0, 0)`.

Vector3 **get_point_position**(idx: int) const

Returns the position of the vertex `idx`. If the index is out of bounds, the function sends an error to the console, and returns `(0, 0, 0)`.

---

float **get_point_tilt**(idx: int) const

Returns the tilt angle in radians for the point `idx`. If the index is out of bounds, the function sends an error to the console, and returns `0`.

---

void **remove_point**(idx: int)

Deletes the point `idx` from the curve. Sends an error to the console if `idx` is out of bounds.

Vector3 **sample**(idx: int, t: float) const

Returns the position between the vertex `idx` and the vertex `idx + 1`, where `t` controls if the point is the first vertex (`t = 0.0`), the last vertex (`t = 1.0`), or in between. Values of `t` outside the range (`0.0 >= t <=1`) give strange, but predictable results.

If `idx` is out of bounds it is truncated to the first or last vertex, and `t` is ignored. If the curve has no points, the function sends an error to the console, and returns `(0, 0, 0)`.

Vector3 **sample_baked**(offset: float = 0.0, cubic: bool = false) const 🔗

Returns a point within the curve at position `offset`, where `offset` is measured as a distance in 3D units along the curve. To do that, it finds the two cached points where the `offset` lies between, then interpolates the values. This interpolation is cubic if `cubic` is set to `true`, or linear if set to `false`.

Cubic interpolation tends to follow the curves better, but linear is faster (and often, precise enough).

Vector3 **sample_baked_up_vector**(offset: float, apply_tilt: bool = false) const 🔗

Returns an up vector within the curve at position `offset`, where `offset` is measured as a distance in 3D units along the curve. To do that, it finds the two cached up vectors where the `offset` lies between, then interpolates the values. If `apply_tilt` is `true`, an interpolated tilt is applied to the interpolated up vector.

If the curve has no up vectors, the function sends an error to the console, and returns `(0, 1, 0)`.

Transform3D **sample_baked_with_rotation**(offset: float = 0.0, cubic: bool = false, apply_tilt: bool = false) const 🔗

Returns a Transform3D with `origin` as point position, `basis.x` as sideway vector, `basis.y` as up vector, `basis.z` as forward vector. When the curve length is 0, there is no reasonable way to calculate the rotation, all vectors aligned with global space axes. See also sample_baked.

Vector3 **samplef**(fofs: float) const

Returns the position at the vertex `fofs`. It calls sample using the integer part of `fofs` as `idx`, and its fractional part as `t`.

void **set_point_in**(idx: int, position: Vector3)

Sets the position of the control point leading to the vertex `idx`. If the index is out of bounds, the function sends an error to the console. The position is relative to the vertex.

void **set_point_out**(idx: int, position: Vector3)

Sets the position of the control point leading out of the vertex `idx`. If the index is out of bounds, the function sends an error to the console. The position is relative to the vertex.

void **set_point_position**(idx: int, position: Vector3)

Sets the position for the vertex `idx`. If the index is out of bounds, the function sends an error to the console.

void **set_point_tilt**(idx: int, tilt: float)

Sets the tilt angle in radians for the point `idx`. If the index is out of bounds, the function sends an error to the console.

The tilt controls the rotation along the look-at axis an object traveling the path would have. In the case of a curve controlling a PathFollow3D, this tilt is an offset over the natural tilt the PathFollow3D calculates.

PackedVector3Array **tessellate**(max_stages: int = 5, tolerance_degrees: float = 4) const 🔗

Returns a list of points along the curve, with a curvature controlled point density. That is, the curvier parts will have more points than the straighter parts.

This approximation makes straight segments between each point, then subdivides those segments until the resulting shape is similar enough.

`max_stages` controls how many subdivisions a curve segment may face before it is considered approximate enough. Each subdivision splits the segment in half, so the default 5 stages may mean up to 32 subdivisions per curve segment. Increase with care!

`tolerance_degrees` controls how many degrees the midpoint of a segment may deviate from the real curve, before the segment has to be subdivided.

PackedVector3Array **tessellate_even_length**(max_stages: int = 5, tolerance_length: float = 0.2) const 🔗

Returns a list of points along the curve, with almost uniform density. `max_stages` controls how many subdivisions a curve segment may face before it is considered approximate enough. Each subdivision splits the segment in half, so the default 5 stages may mean up to 32 subdivisions per curve segment. Increase with care!

`tolerance_length` controls the maximal distance between two neighboring points, before the segment has to be subdivided.

CurveTexture

Inherits: Texture2D **<** Texture **<** Resource **<** RefCounted **<** Object

A 1D texture where pixel brightness corresponds to points on a curve.

Description

A 1D texture where pixel brightness corresponds to points on a Curve resource, either in grayscale or in red. This visual representation simplifies the task of saving curves as image files.

If you need to store up to 3 curves within a single texture, use CurveXYZTexture instead. See also GradientTexture1D and GradientTexture2D.

Properties

Curve	curve	
bool	resource_local_to_scene	`false` (overrides Resource)
TextureMode	texture_mode	`0`
int	width	`256`

Enumerations

enum **TextureMode**:

TextureMode **TEXTURE_MODE_RGB** = `0`

Store the curve equally across the red, green and blue channels. This uses more video memory, but is more compatible with shaders that only read the green and blue values.

TextureMode **TEXTURE_MODE_RED** = `1`

Store the curve only in the red channel. This saves video memory, but some custom shaders may not be able to work with this.

Property Descriptions

Curve **curve**

- void **set_curve**(value: Curve)
- Curve **get_curve**()

The Curve that is rendered onto the texture.

TextureMode **texture_mode** = `0`

- void **set_texture_mode**(value: TextureMode)
- TextureMode **get_texture_mode**()

The format the texture should be generated with. When passing a CurveTexture as an input to a Shader, this may need to be adjusted.

---

int **width** = `256` 🔗

- void **set_width**(value: int)
- int **get_width**()

The width of the texture (in pixels). Higher values make it possible to represent high-frequency data better (such as sudden direction changes), at the cost of increased generation time and memory usage.

CurveXYZTexture

Inherits: Texture2D **<** Texture **<** Resource **<** RefCounted **<** Object

A 1D texture where the red, green, and blue color channels correspond to points on 3 curves.

Description

A 1D texture where the red, green, and blue color channels correspond to points on 3 Curve resources. Compared to using separate CurveTextures, this further simplifies the task of saving curves as image files.

If you only need to store one curve within a single texture, use CurveTexture instead. See also GradientTexture1D and GradientTexture2D.

Properties

Curve	curve_x	
Curve	curve_y	
Curve	curve_z	
bool	resource_local_to_scene	`false` (overrides Resource)

int	width	`256`

Property Descriptions

Curve **curve_x**

- void **set_curve_x**(value: Curve)
- Curve **get_curve_x**()

The Curve that is rendered onto the texture's red channel.

Curve **curve_y**

- void **set_curve_y**(value: Curve)
- Curve **get_curve_y**()

The Curve that is rendered onto the texture's green channel.

Curve **curve_z**

- void **set_curve_z**(value: Curve)
- Curve **get_curve_z**()

The Curve that is rendered onto the texture's blue channel.

int **width** = `256`

- void **set_width**(value: int)
- int **get_width**()

The width of the texture (in pixels). Higher values make it possible to represent high-frequency data better (such as sudden direction changes), at the cost of increased generation time and memory usage.

CylinderMesh

Inherits: PrimitiveMesh **<** Mesh **<** Resource **<** RefCounted **<** Object

Class representing a cylindrical PrimitiveMesh.

Description

Class representing a cylindrical PrimitiveMesh. This class can be used to create cones by setting either the top_radius or bottom_radius properties to `0.0`.

Properties

float	bottom_radius	`0.5`
bool	cap_bottom	`true`
bool	cap_top	`true`
float	height	`2.0`
int	radial_segments	`64`
int	rings	`4`

float	top_radius	`0.5`

Property Descriptions

float **bottom_radius** = `0.5`

- void **set_bottom_radius**(value: float)
- float **get_bottom_radius**()

Bottom radius of the cylinder. If set to `0.0`, the bottom faces will not be generated, resulting in a conic shape. See also cap_bottom.

bool **cap_bottom** = `true`

- void **set_cap_bottom**(value: bool)
- bool **is_cap_bottom**()

If `true`, generates a cap at the bottom of the cylinder. This can be set to `false` to speed up generation and rendering when the cap is never seen by the camera. See also bottom_radius.

Note: If bottom_radius is `0.0`, cap generation is always skipped even if cap_bottom is `true`.

bool **cap_top** = `true`

- void **set_cap_top**(value: bool)
- bool **is_cap_top**()

If `true`, generates a cap at the top of the cylinder. This can be set to `false` to speed up generation and rendering when the cap is never seen by the camera. See also top_radius.

Note: If top_radius is `0.0`, cap generation is always skipped even if cap_top is `true`.

---

float **height** = `2.0`

- void **set_height**(value: float)
- float **get_height**()

Full height of the cylinder.

---

int **radial_segments** = `64`

- void **set_radial_segments**(value: int)
- int **get_radial_segments**()

Number of radial segments on the cylinder. Higher values result in a more detailed cylinder/cone at the cost of performance.

---

int **rings** = `4`

- void **set_rings**(value: int)
- int **get_rings**()

Number of edge rings along the height of the cylinder. Changing rings does not have any visual impact unless a shader or procedural mesh tool is used to alter the vertex data. Higher values result in more subdivisions, which can be used to create smoother-looking effects with shaders or procedural mesh tools (at the cost of performance). When not altering the vertex data using a shader or procedural mesh tool, rings should be kept to its default value.

---

float **top_radius** = `0.5` 🔗

- void **set_top_radius**(value: float)
- float **get_top_radius**()

Top radius of the cylinder. If set to `0.0`, the top faces will not be generated, resulting in a conic shape. See also cap_top.

CylinderShape3D

Inherits: Shape3D **<** Resource **<** RefCounted **<** Object

A 3D cylinder shape used for physics collision.

Description

A 3D cylinder shape, intended for use in physics. Usually used to provide a shape for a CollisionShape3D.

Note: There are several known bugs with cylinder collision shapes. Using CapsuleShape3D or BoxShape3D instead is recommended.

Performance: CylinderShape3D is fast to check collisions against, but it is slower than CapsuleShape3D, BoxShape3D, and SphereShape3D.

Tutorials

- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]
- 3D Physics Tests Demo [https://godotengine.org/asset-library/asset/2747]
- 3D Voxel Demo [https://godotengine.org/asset-library/asset/2755]

Properties

float	height	`2.0`

float	radius	`0.5`

Property Descriptions

float **height** = `2.0` 🔗

- void **set_height**(value: float)
- float **get_height**()

The cylinder's height.

---

float **radius** = `0.5` 🔗

- void **set_radius**(value: float)
- float **get_radius**()

The cylinder's radius.

EditorNode3DGizmoPlugin

Inherits: Resource **<** RefCounted **<** Object

A class used by the editor to define Node3D gizmo types.

Description

EditorNode3DGizmoPlugin allows you to define a new type of Gizmo. There are two main ways to do so: extending **EditorNode3DGizmoPlugin** for the simpler gizmos, or creating a new EditorNode3DGizmo type. See the tutorial in the documentation for more info.

To use **EditorNode3DGizmoPlugin**, register it using the EditorPlugin.add_node_3d_gizmo_plugin method first.

Tutorials

- Node3D gizmo plugins

Methods

void	_begin_handle_action(gizmo: EditorNode3DGizmo, handle_id: int, secondary: bool) virtual
bool	_can_be_hidden() virtual const

void	_commit_handle(gizmo: EditorNode3DGizmo, handle_id: int, secondary: bool, restore: Variant, cancel: bool) virtual
void	_commit_subgizmos(gizmo: EditorNode3DGizmo, ids: PackedInt32Array, restores: Array[Transform3D], cancel: bool) virtual
EditorNode3DGizmo	_create_gizmo(for_node_3d: Node3D) virtual const
String	_get_gizmo_name() virtual const
String	_get_handle_name(gizmo: EditorNode3DGizmo, handle_id: int, secondary: bool) virtual const
Variant	_get_handle_value(gizmo: EditorNode3DGizmo, handle_id: int, secondary: bool) virtual const
int	_get_priority() virtual const
Transform3D	_get_subgizmo_transform(gizmo: EditorNode3DGizmo, subgizmo_id: int) virtual const

bool	_has_gizmo(for_node_3d: Node3D) virtual const
bool	_is_handle_highlighted(gizmo: EditorNode3DGizmo, handle_id: int, secondary: bool) virtual const
bool	_is_selectable_when_hidden() virtual const
void	_redraw(gizmo: EditorNode3DGizmo) virtual
void	_set_handle(gizmo: EditorNode3DGizmo, handle_id: int, secondary: bool, camera: Camera3D, screen_pos: Vector2) virtual
void	_set_subgizmo_transform(gizmo: EditorNode3DGizmo, subgizmo_id: int, transform: Transform3D) virtual
PackedInt32Array	_subgizmos_intersect_frustum(gizmo: EditorNode3DGizmo, camera: Camera3D, frustum_planes: Array[Plane]) virtual const
int	_subgizmos_intersect_ray(gizmo: EditorNode3DGizmo, camera: Camera3D, screen_pos: Vector2) virtual const

void	add_material(name: String, material: StandardMaterial3D)
void	create_handle_material(name: String, billboard: bool = false, texture: Texture2D = null)
void	create_icon_material(name: String, texture: Texture2D, on_top: bool = false, color: Color = Color(1, 1, 1, 1))
void	create_material(name: String, color: Color, billboard: bool = false, on_top: bool = false, use_vertex_color: bool = false)
StandardMaterial3D	get_material(name: String, gizmo: EditorNode3DGizmo = null)

Method Descriptions

void **_begin_handle_action**(gizmo: EditorNode3DGizmo, handle_id: int, secondary: bool) virtual 🔗

There is currently no description for this method. Please help us by contributing one!

---

bool **_can_be_hidden**() virtual const 🔗

Override this method to define whether the gizmos handled by this plugin can be hidden or not. Returns `true` if not overridden.

---

void **_commit_handle**(gizmo: EditorNode3DGizmo, handle_id: int, secondary: bool, restore: Variant, cancel: bool) virtual 🔗

Override this method to commit a handle being edited (handles must have been previously added by EditorNode3DGizmo.add_handles during _redraw). This usually means creating an UndoRedo action for the change, using the current handle value as "do" and the `restore` argument as "undo".

If the `cancel` argument is `true`, the `restore` value should be directly set, without any UndoRedo action.

The `secondary` argument is `true` when the committed handle is secondary (see EditorNode3DGizmo.add_handles for more information).

Called for this plugin's active gizmos.

---

void **_commit_subgizmos**(gizmo: EditorNode3DGizmo, ids: PackedInt32Array, restores: Array[Transform3D], cancel: bool) virtual 🔗

Override this method to commit a group of subgizmos being edited (see _subgizmos_intersect_ray and _subgizmos_intersect_frustum). This usually means creating an UndoRedo action for the change, using the current transforms as "do" and the `restores` transforms as "undo".

If the `cancel` argument is `true`, the `restores` transforms should be directly set, without any UndoRedo action. As with all subgizmo methods, transforms are given in local space respect to the gizmo's Node3D. Called for this plugin's active gizmos.

EditorNode3DGizmo **_create_gizmo**(for_node_3d: Node3D) virtual const

Override this method to return a custom EditorNode3DGizmo for the spatial nodes of your choice, return `null` for the rest of nodes. See also _has_gizmo.

String **_get_gizmo_name**() virtual const

Override this method to provide the name that will appear in the gizmo visibility menu.

String **_get_handle_name**(gizmo: EditorNode3DGizmo, handle_id: int, secondary: bool) virtual const

Override this method to provide gizmo's handle names. The `secondary` argument is `true` when the requested handle is secondary (see EditorNode3DGizmo.add_handles for more information). Called for this plugin's active gizmos.

Variant **_get_handle_value**(gizmo: EditorNode3DGizmo, handle_id: int, secondary: bool) virtual const

Override this method to return the current value of a handle. This value will be requested at the start of an edit and used as the `restore` argument in _commit_handle.

The `secondary` argument is `true` when the requested handle is secondary (see EditorNode3DGizmo.add_handles for more information).

Called for this plugin's active gizmos.

---

int **_get_priority**() virtual const

Override this method to set the gizmo's priority. Gizmos with higher priority will have precedence when processing inputs like handles or subgizmos selection.

All built-in editor gizmos return a priority of `-1`. If not overridden, this method will return `0`, which means custom gizmos will automatically get higher priority than built-in gizmos.

Transform3D **_get_subgizmo_transform**(gizmo: EditorNode3DGizmo, subgizmo_id: int) virtual const

Override this method to return the current transform of a subgizmo. As with all subgizmo methods, the transform should be in local space respect to the gizmo's Node3D. This transform will be requested at the start of an edit and used in the `restore` argument in _commit_subgizmos. Called for this plugin's active gizmos.

---

bool **_has_gizmo**(for_node_3d: Node3D) virtual const

Override this method to define which Node3D nodes have a gizmo from this plugin. Whenever a Node3D node is added to a scene this method is called, if it returns `true` the node gets a generic EditorNode3DGizmo assigned and is added to this plugin's list of active gizmos.

---

bool **_is_handle_highlighted**(gizmo: EditorNode3DGizmo, handle_id: int, secondary: bool) virtual const 🔗

Override this method to return `true` whenever to given handle should be highlighted in the editor. The `secondary` argument is `true` when the requested handle is secondary (see EditorNode3DGizmo.add_handles for more information). Called for this plugin's active gizmos.

---

bool **_is_selectable_when_hidden**() virtual const 🔗

Override this method to define whether Node3D with this gizmo should be selectable even when the gizmo is hidden.

---

void **_redraw**(gizmo: EditorNode3DGizmo) virtual 🔗

Override this method to add all the gizmo elements whenever a gizmo update is requested. It's common to call EditorNode3DGizmo.clear at the beginning of this method and then add visual elements depending on the node's properties.

---

void **_set_handle**(gizmo: EditorNode3DGizmo, handle_id: int, secondary: bool, camera: Camera3D, screen_pos: Vector2) virtual 🔗

Override this method to update the node's properties when the user drags a gizmo handle (previously added with EditorNode3DGizmo.add_handles). The provided `screen_pos` is the mouse position in screen coordinates and the `camera` can be used to convert it to raycasts.

The `secondary` argument is `true` when the edited handle is secondary (see EditorNode3DGizmo.add_handles for more information).

Called for this plugin's active gizmos.

void **_set_subgizmo_transform**(gizmo: EditorNode3DGizmo, subgizmo_id: int, transform: Transform3D) virtual 🔗

Override this method to update the node properties during subgizmo editing (see _subgizmos_intersect_ray and _subgizmos_intersect_frustum). The `transform` is given in the Node3D's local coordinate system. Called for this plugin's active gizmos.

PackedInt32Array **_subgizmos_intersect_frustum**(gizmo: EditorNode3DGizmo, camera: Camera3D, frustum_planes: Array[Plane]) virtual const 🔗

Override this method to allow selecting subgizmos using mouse drag box selection. Given a `camera` and `frustum_planes`, this method should return which subgizmos are contained within the frustums. The `frustum_planes` argument consists of an array with all the Planes that make up the selection frustum. The returned value should contain a list of unique subgizmo identifiers, these identifiers can have any non-negative value and will be used in other virtual methods like _get_subgizmo_transform or _commit_subgizmos. Called for this plugin's active gizmos.

int **_subgizmos_intersect_ray**(gizmo: EditorNode3DGizmo, camera: Camera3D, screen_pos: Vector2) virtual const 🔗

Override this method to allow selecting subgizmos using mouse clicks. Given a `camera` and a `screen_pos` in screen coordinates, this method should return which subgizmo should be selected. The returned value should be a unique subgizmo identifier, which can have any non-negative value and will be used in other virtual methods like _get_subgizmo_transform or _commit_subgizmos. Called for this plugin's active gizmos.

void **add_material**(name: String, material: StandardMaterial3D) 🔗

Adds a new material to the internal material list for the plugin. It can then be accessed with get_material. Should not be overridden.

void **create_handle_material**(name: String, billboard: bool = false, texture: Texture2D = null) 🔗

Creates a handle material with its variants (selected and/or editable) and adds them to the internal material list. They can then be accessed with get_material and used in EditorNode3DGizmo.add_handles. Should not be overridden.

You can optionally provide a texture to use instead of the default icon.

void **create_icon_material**(name: String, texture: Texture2D, on_top: bool = false, color: Color = Color(1, 1, 1, 1)) 🔗

Creates an icon material with its variants (selected and/or editable) and adds them to the internal material list. They can then be accessed with get_material and used in EditorNode3DGizmo.add_unscaled_billboard. Should not be overridden.

void **create_material**(name: String, color: Color, billboard: bool = false, on_top: bool = false, use_vertex_color: bool = false) 🔗

Creates an unshaded material with its variants (selected and/or editable) and adds them to the internal material list. They can then be accessed with get_material and used in EditorNode3DGizmo.add_mesh and EditorNode3DGizmo.add_lines. Should not be overridden.

StandardMaterial3D **get_material**(name: String, gizmo: EditorNode3DGizmo = null) 🔗

Gets material from the internal list of materials. If an EditorNode3DGizmo is provided, it will try to get the corresponding variant (selected and/or editable).

EditorSettings

Inherits: Resource **<** RefCounted **<** Object

Object that holds the project-independent editor settings.

Description

Object that holds the project-independent editor settings. These settings are generally visible in the **Editor > Editor Settings** menu.

Property names use slash delimiters to distinguish sections. Setting values can be of any Variant type. It's recommended to use `snake_case` for editor settings to be consistent with the Godot editor itself.

Accessing the settings can be done using the following methods, such as:

GDScript

```
var settings = EditorInterface.get_editor_settings()
# `settings.set("some/property", 10)` also works as this class overrides `_set()` internally.
settings.set_setting("some/property", 10)
# `settings.get("some/property")` also works as this class overrides `_get()` internally.
settings.get_setting("some/property")
var list_of_settings = settings.get_property_list()
```

C#

```
EditorSettings settings = EditorInterface.Singleton.GetEditorSettings();
// `settings.set("some/property", value)` also works as this class overrides `_set()`
internally.
settings.SetSetting("some/property", Value);
// `settings.get("some/property", value)` also works as this class overrides `_get()`
internally.
settings.GetSetting("some/property");
Godot.Collections.Array<Godot.Collections.Dictionary> listOfSettings =
settings.GetPropertyList();
```

Note: This class shouldn't be instantiated directly. Instead, access the singleton using EditorInterface.get_editor_settings.

Properties

bool	debugger/auto_switch_to_remote_scene_tree
bool	debugger/profile_native_calls
int	debugger/profiler_frame_history_size
int	debugger/profiler_frame_max_functions

float	debugger/remote_inspect_refresh_interval
float	debugger/remote_scene_tree_refresh_interval
bool	docks/filesystem/always_show_folders
String	docks/filesystem/textfile_extensions
int	docks/filesystem/thumbnail_size
float	docks/property_editor/auto_refresh_interval
float	docks/property_editor/subresource_hue_tint
bool	docks/scene_tree/auto_expand_to_selected
bool	docks/scene_tree/center_node_on_reparent
bool	docks/scene_tree/start_create_dialog_fully_expanded
Color	editors/2d/bone_color1
Color	editors/2d/bone_color2
Color	editors/2d/bone_ik_color
Color	editors/2d/bone_outline_color
float	editors/2d/bone_outline_size
Color	editors/2d/bone_selected_color
float	editors/2d/bone_width
Color	editors/2d/grid_color
Color	editors/2d/guides_color

Color	editors/2d/smart_snapping_line_color
bool	editors/2d/use_integer_zoom_by_default
Color	editors/2d/viewport_border_color
float	editors/3d/default_fov
float	editors/3d/default_z_far
float	editors/3d/default_z_near
int	editors/3d/freelook/freelook_activation_modifier
float	editors/3d/freelook/freelook_base_speed
float	editors/3d/freelook/freelook_inertia
int	editors/3d/freelook/freelook_navigation_scheme
float	editors/3d/freelook/freelook_sensitivity
bool	editors/3d/freelook/freelook_speed_zoom_link
float	editors/3d/grid_division_level_bias
int	editors/3d/grid_division_level_max
int	editors/3d/grid_division_level_min
int	editors/3d/grid_size
bool	editors/3d/grid_xy_plane
bool	editors/3d/grid_xz_plane
bool	editors/3d/grid_yz_plane

bool	editors/3d/navigation/emulate_3_button_mouse
bool	editors/3d/navigation/emulate_numpad
bool	editors/3d/navigation/invert_x_axis
bool	editors/3d/navigation/invert_y_axis
int	editors/3d/navigation/navigation_scheme
int	editors/3d/navigation/orbit_modifier
int	editors/3d/navigation/pan_modifier
bool	editors/3d/navigation/warped_mouse_panning
int	editors/3d/navigation/zoom_modifier
int	editors/3d/navigation/zoom_style
float	editors/3d/navigation_feel/orbit_inertia
float	editors/3d/navigation_feel/orbit_sensitivity
float	editors/3d/navigation_feel/translation_inertia
float	editors/3d/navigation_feel/zoom_inertia
Color	editors/3d/primary_grid_color
int	editors/3d/primary_grid_steps
Color	editors/3d/secondary_grid_color
Color	editors/3d/selection_box_color
Color	editors/3d_gizmos/gizmo_colors/aabb

Color	editors/3d_gizmos/gizmo_colors/instantiated
Color	editors/3d_gizmos/gizmo_colors/joint
bool	editors/animation/autorename_animation_tracks
bool	editors/animation/confirm_insert_track
bool	editors/animation/default_create_bezier_tracks
bool	editors/animation/default_create_reset_tracks
Color	editors/animation/onion_layers_future_color
Color	editors/animation/onion_layers_past_color
float	editors/grid_map/pick_distance
int	editors/panning/2d_editor_pan_speed
int	editors/panning/2d_editor_panning_scheme
int	editors/panning/animation_editors_panning_scheme
bool	editors/panning/simple_panning
int	editors/panning/sub_editors_panning_scheme
bool	editors/panning/warped_mouse_panning
float	editors/polygon_editor/auto_bake_delay
int	editors/polygon_editor/point_grab_radius
bool	editors/polygon_editor/show_previous_outline
bool	editors/shader_editor/behavior/files/restore_shaders_on_load

bool	editors/tiles_editor/display_grid
Color	editors/tiles_editor/grid_color
bool	editors/tiles_editor/highlight_selected_layer
Color	editors/visual_editors/category_colors/color_color
Color	editors/visual_editors/category_colors/conditional_color
Color	editors/visual_editors/category_colors/input_color
Color	editors/visual_editors/category_colors/output_color
Color	editors/visual_editors/category_colors/particle_color
Color	editors/visual_editors/category_colors/scalar_color
Color	editors/visual_editors/category_colors/special_color
Color	editors/visual_editors/category_colors/textures_color
Color	editors/visual_editors/category_colors/transform_color
Color	editors/visual_editors/category_colors/utility_color
Color	editors/visual_editors/category_colors/vector_color
String	editors/visual_editors/color_theme
Color	editors/visual_editors/connection_colors/boolean_color
Color	editors/visual_editors/connection_colors/sampler_color
Color	editors/visual_editors/connection_colors/scalar_color
Color	editors/visual_editors/connection_colors/transform_color

Color	editors/visual_editors/connection_colors/vector2_color
Color	editors/visual_editors/connection_colors/vector3_color
Color	editors/visual_editors/connection_colors/vector4_color
int	editors/visual_editors/grid_pattern
float	editors/visual_editors/lines_curvature
float	editors/visual_editors/minimap_opacity
int	editors/visual_editors/visual_shader/port_preview_size
String	filesystem/directories/autoscan_project_path
String	filesystem/directories/default_project_path
String	filesystem/external_programs/3d_model_editor
String	filesystem/external_programs/audio_editor
String	filesystem/external_programs/raster_image_editor
String	filesystem/external_programs/terminal_emulator
String	filesystem/external_programs/terminal_emulator_flags
String	filesystem/external_programs/vector_image_editor
int	filesystem/file_dialog/display_mode
bool	filesystem/file_dialog/show_hidden_files
int	filesystem/file_dialog/thumbnail_size
String	filesystem/import/blender/blender_path

int	filesystem/import/blender/rpc_port
float	filesystem/import/blender/rpc_server_uptime
String	filesystem/import/fbx/fbx2gltf_path
bool	filesystem/on_save/compress_binary_resources
bool	filesystem/on_save/safe_save_on_backup_then_rename
String	filesystem/tools/oidn/oidn_denoise_path
bool	input/buffering/agile_event_flushing
bool	input/buffering/use_accumulated_input
int	interface/editor/accept_dialog_cancel_ok_buttons
bool	interface/editor/automatically_open_screenshots
String	interface/editor/code_font
int	interface/editor/code_font_contextual_ligatures
String	interface/editor/code_font_custom_opentype_features
String	interface/editor/code_font_custom_variations
int	interface/editor/code_font_size
float	interface/editor/custom_display_scale
int	interface/editor/display_scale
int	interface/editor/dock_tab_style
String	interface/editor/editor_language

int	interface/editor/editor_screen
bool	interface/editor/expand_to_title
int	interface/editor/font_antialiasing
bool	interface/editor/font_disable_embedded_bitmaps
int	interface/editor/font_hinting
int	interface/editor/font_subpixel_positioning
bool	interface/editor/import_resources_when_unfocused
bool	interface/editor/localize_settings
int	interface/editor/low_processor_mode_sleep_usec
String	interface/editor/main_font
String	interface/editor/main_font_bold
int	interface/editor/main_font_size
bool	interface/editor/mouse_extra_buttons_navigate_history
int	interface/editor/project_manager_screen
bool	interface/editor/save_each_scene_on_quit
bool	interface/editor/save_on_focus_loss
bool	interface/editor/separate_distraction_mode
int	interface/editor/show_internal_errors_in_toast_notifications
int	interface/editor/show_update_spinner

bool	interface/editor/single_window_mode
int	interface/editor/ui_layout_direction
int	interface/editor/unfocused_low_processor_mode_sleep_usec
bool	interface/editor/update_continuously
bool	interface/editor/use_embedded_menu
bool	interface/editor/use_native_file_dialogs
int	interface/editor/vsync_mode
bool	interface/inspector/auto_unfold_foreign_scenes
int	interface/inspector/default_color_picker_mode
int	interface/inspector/default_color_picker_shape
float	interface/inspector/default_float_step
int	interface/inspector/default_property_name_style
bool	interface/inspector/delimitate_all_container_and_resources
bool	interface/inspector/disable_folding
float	interface/inspector/float_drag_speed
bool	interface/inspector/horizontal_vector2_editing
bool	interface/inspector/horizontal_vector_types_editing
int	interface/inspector/max_array_dictionary_items_per_page
int	interface/inspector/nested_color_mode

bool	interface/inspector/open_resources_in_current_inspector
PackedStringArray	interface/inspector/resources_to_open_in_new_inspector
bool	interface/inspector/show_low_level_opentype_features
bool	interface/multi_window/enable
bool	interface/multi_window/maximize_window
bool	interface/multi_window/restore_windows_on_load
int	interface/scene_tabs/display_close_button
int	interface/scene_tabs/maximum_width
bool	interface/scene_tabs/restore_scenes_on_load
bool	interface/scene_tabs/show_script_button
bool	interface/scene_tabs/show_thumbnail_on_hover
Color	interface/theme/accent_color
int	interface/theme/additional_spacing
Color	interface/theme/base_color
int	interface/theme/base_spacing
int	interface/theme/border_size
float	interface/theme/contrast
int	interface/theme/corner_radius
String	interface/theme/custom_theme

bool	interface/theme/draw_extra_borders
bool	interface/theme/follow_system_theme
int	interface/theme/icon_and_font_color
float	interface/theme/icon_saturation
String	interface/theme/preset
float	interface/theme/relationship_line_opacity
String	interface/theme/spacing_preset
bool	interface/theme/use_system_accent_color
bool	interface/touchscreen/enable_long_press_as_right_click
bool	interface/touchscreen/enable_pan_and_scale_gestures
bool	interface/touchscreen/increase_scrollbar_touch_area
float	interface/touchscreen/scale_gizmo_handles
int	network/connection/engine_version_update_mode
int	network/connection/network_mode
String	network/debug/remote_host
int	network/debug/remote_port
String	network/http_proxy/host
int	network/http_proxy/port
String	network/tls/editor_tls_certificates

String	project_manager/default_renderer
int	project_manager/directory_naming_convention
int	project_manager/sorting_order
bool	run/auto_save/save_before_running
int	run/bottom_panel/action_on_play
int	run/bottom_panel/action_on_stop
bool	run/output/always_clear_output_on_play
int	run/output/font_size
int	run/output/max_lines
bool	run/platforms/linuxbsd/prefer_wayland
int	run/window_placement/android_window
int	run/window_placement/rect
Vector2	run/window_placement/rect_custom_position
int	run/window_placement/screen
bool	text_editor/appearance/caret/caret_blink
float	text_editor/appearance/caret/caret_blink_interval
bool	text_editor/appearance/caret/highlight_all_occurrences
bool	text_editor/appearance/caret/highlight_current_line
int	text_editor/appearance/caret/type

int	text_editor/appearance/guidelines/line_length_guideline_hard_column
int	text_editor/appearance/guidelines/line_length_guideline_soft_column
bool	text_editor/appearance/guidelines/show_line_length_guidelines
bool	text_editor/appearance/gutters/highlight_type_safe_lines
bool	text_editor/appearance/gutters/line_numbers_zero_padded
bool	text_editor/appearance/gutters/show_info_gutter
bool	text_editor/appearance/gutters/show_line_numbers
int	text_editor/appearance/lines/autowrap_mode
bool	text_editor/appearance/lines/code_folding
int	text_editor/appearance/lines/word_wrap
int	text_editor/appearance/minimap/minimap_width
bool	text_editor/appearance/minimap/show_minimap
bool	text_editor/appearance/whitespace/draw_spaces
bool	text_editor/appearance/whitespace/draw_tabs
int	text_editor/appearance/whitespace/line_spacing
bool	text_editor/behavior/files/auto_reload_scripts_on_external_change
int	text_editor/behavior/files/autosave_interval_secs
bool	text_editor/behavior/files/convert_indent_on_save
bool	text_editor/behavior/files/restore_scripts_on_load

bool	text_editor/behavior/files/trim_final_newlines_on_save
bool	text_editor/behavior/files/trim_trailing_whitespace_on_save
bool	text_editor/behavior/indent/auto_indent
bool	text_editor/behavior/indent/indent_wrapped_lines
int	text_editor/behavior/indent/size
int	text_editor/behavior/indent/type
String	text_editor/behavior/navigation/custom_word_separators
bool	text_editor/behavior/navigation/drag_and_drop_selection
bool	text_editor/behavior/navigation/move_caret_on_right_click
bool	text_editor/behavior/navigation/open_script_when_connecting_signal_to_existing_me
bool	text_editor/behavior/navigation/scroll_past_end_of_file
bool	text_editor/behavior/navigation/smooth_scrolling
bool	text_editor/behavior/navigation/stay_in_script_editor_on_node_selected
bool	text_editor/behavior/navigation/use_custom_word_separators
bool	text_editor/behavior/navigation/use_default_word_separators
int	text_editor/behavior/navigation/v_scroll_speed
bool	text_editor/completion/add_node_path_literals
bool	text_editor/completion/add_string_name_literals
bool	text_editor/completion/add_type_hints

bool	text_editor/completion/auto_brace_complete
float	text_editor/completion/code_complete_delay
bool	text_editor/completion/code_complete_enabled
bool	text_editor/completion/colorize_suggestions
bool	text_editor/completion/complete_file_paths
float	text_editor/completion/idle_parse_delay
bool	text_editor/completion/put_callhint_tooltip_below_current_line
bool	text_editor/completion/use_single_quotes
int	text_editor/help/class_reference_examples
int	text_editor/help/help_font_size
int	text_editor/help/help_source_font_size
int	text_editor/help/help_title_font_size
bool	text_editor/help/show_help_index
bool	text_editor/script_list/show_members_overview
bool	text_editor/script_list/sort_members_outline_alphabetically
String	text_editor/theme/color_theme
Color	text_editor/theme/highlighting/background_color
Color	text_editor/theme/highlighting/base_type_color
Color	text_editor/theme/highlighting/bookmark_color

Color	text_editor/theme/highlighting/brace_mismatch_color
Color	text_editor/theme/highlighting/breakpoint_color
Color	text_editor/theme/highlighting/caret_background_color
Color	text_editor/theme/highlighting/caret_color
Color	text_editor/theme/highlighting/code_folding_color
Color	text_editor/theme/highlighting/comment_color
Color	text_editor/theme/highlighting/completion_background_color
Color	text_editor/theme/highlighting/completion_existing_color
Color	text_editor/theme/highlighting/completion_font_color
Color	text_editor/theme/highlighting/completion_scroll_color
Color	text_editor/theme/highlighting/completion_scroll_hovered_color
Color	text_editor/theme/highlighting/completion_selected_color
Color	text_editor/theme/highlighting/control_flow_keyword_color
Color	text_editor/theme/highlighting/current_line_color
Color	text_editor/theme/highlighting/doc_comment_color
Color	text_editor/theme/highlighting/engine_type_color
Color	text_editor/theme/highlighting/executing_line_color
Color	text_editor/theme/highlighting/folded_code_region_color
Color	text_editor/theme/highlighting/function_color

Color	text_editor/theme/highlighting/keyword_color
Color	text_editor/theme/highlighting/line_length_guideline_color
Color	text_editor/theme/highlighting/line_number_color
Color	text_editor/theme/highlighting/mark_color
Color	text_editor/theme/highlighting/member_variable_color
Color	text_editor/theme/highlighting/number_color
Color	text_editor/theme/highlighting/safe_line_number_color
Color	text_editor/theme/highlighting/search_result_border_color
Color	text_editor/theme/highlighting/search_result_color
Color	text_editor/theme/highlighting/selection_color
Color	text_editor/theme/highlighting/string_color
Color	text_editor/theme/highlighting/symbol_color
Color	text_editor/theme/highlighting/text_color
Color	text_editor/theme/highlighting/text_selected_color
Color	text_editor/theme/highlighting/user_type_color
Color	text_editor/theme/highlighting/word_highlighted_color

Methods

void	add_property_info(info: Dictionary)
bool	check_changed_settings_in_group(setting_prefix: String) const

void	erase(property: String)
PackedStringArray	get_changed_settings() const
PackedStringArray	get_favorites() const
Variant	get_project_metadata(section: String, key: String, default: Variant = null) const
PackedStringArray	get_recent_dirs() const
Variant	get_setting(name: String) const
bool	has_setting(name: String) const
void	mark_setting_changed(setting: String)
void	set_builtin_action_override(name: String, actions_list: Array[InputEvent])
void	set_favorites(dirs: PackedStringArray)
void	set_initial_value(name: StringName, value: Variant, update_current: bool)
void	set_project_metadata(section: String, key: String, data: Variant)
void	set_recent_dirs(dirs: PackedStringArray)
void	set_setting(name: String, value: Variant)

Signals

settings_changed() 🔗

Emitted after any editor setting has changed.

Constants

NOTIFICATION_EDITOR_SETTINGS_CHANGED = `10000` 🔗

Emitted after any editor setting has changed. It's used by various editor plugins to update their visuals on theme changes or logic on configuration changes.

Property Descriptions

bool **debugger/auto_switch_to_remote_scene_tree** 🔗

If `true`, automatically switches to the **Remote** scene tree when running the project from the editor. If `false`, stays on the **Local** scene tree when running the project from the editor.

---

bool **debugger/profile_native_calls** 🔗

If `true`, enables collection of profiling data from non-GDScript Godot functions, such as engine class methods. Enabling this slows execution while profiling further.

---

int **debugger/profiler_frame_history_size** 🔗

The size of the profiler's frame history. The default value (3600) allows seeing up to 60 seconds of profiling if the project renders at a constant 60 FPS. Higher values allow viewing longer periods of profiling in the graphs, especially when the project is running at high framerates.

---

int **debugger/profiler_frame_max_functions** 🔗

The maximum number of script functions that can be displayed per frame in the profiler. If there are more script functions called in a given profiler frame, these functions will be discarded from the profiling results entirely.

Note: This setting is only read when the profiler is first started, so changing it during profiling will have no effect.

---

float **debugger/remote_inspect_refresh_interval** 🔗

The refresh interval for the remote inspector's properties (in seconds). Lower values are more reactive, but may cause stuttering while the project is running from the editor and the **Remote** scene tree is selected in the Scene tree dock.

---

float **debugger/remote_scene_tree_refresh_interval** 🔗

The refresh interval for the remote scene tree (in seconds). Lower values are more reactive, but may cause stuttering while the project is running from the editor and the **Remote** scene tree is selected in the Scene tree dock.

---

bool **docks/filesystem/always_show_folders** 🔗

If `true`, displays folders in the FileSystem dock's bottom pane when split mode is enabled. If `false`, only files will be displayed in the bottom pane. Split mode can be toggled by pressing the icon next to the `res://` folder path.

Note: This setting has no effect when split mode is disabled (which is the default).

String **docks/filesystem/textfile_extensions**

List of file extensions to consider as editable text files in the FileSystem dock (by double-clicking on the files).

---

int **docks/filesystem/thumbnail_size**

The thumbnail size to use in the FileSystem dock (in pixels). See also filesystem/file_dialog/thumbnail_size.

---

float **docks/property_editor/auto_refresh_interval**

The refresh interval to use for the Inspector dock's properties. The effect of this setting is mainly noticeable when adjusting gizmos in the 2D/3D editor and looking at the inspector at the same time. Lower values make the inspector refresh more often, but take up more CPU time.

---

float **docks/property_editor/subresource_hue_tint**

The tint intensity to use for the subresources background in the Inspector dock. The tint is used to distinguish between different subresources in the inspector. Higher values result in a more noticeable background color difference.

---

bool **docks/scene_tree/auto_expand_to_selected**

If `true`, the scene tree dock will automatically unfold nodes when a node that has folded parents is selected.

---

bool **docks/scene_tree/center_node_on_reparent**

If `true`, new node created when reparenting node(s) will be positioned at the average position of the selected node(s).

---

bool **docks/scene_tree/start_create_dialog_fully_expanded**

If `true`, the Create dialog (Create New Node/Create New Resource) will start with all its sections expanded. Otherwise, sections will be collapsed until the user starts searching (which will automatically expand sections as needed).

Color **editors/2d/bone_color1**

The "start" stop of the color gradient to use for bones in the 2D skeleton editor.

Color **editors/2d/bone_color2**

The "end" stop of the color gradient to use for bones in the 2D skeleton editor.

Color **editors/2d/bone_ik_color**

The color to use for inverse kinematics-enabled bones in the 2D skeleton editor.

Color **editors/2d/bone_outline_color**

The outline color to use for non-selected bones in the 2D skeleton editor. See also editors/2d/bone_selected_color.

---

float **editors/2d/bone_outline_size**

The outline size in the 2D skeleton editor (in pixels). See also editors/2d/bone_width.

Note: Changes to this value only apply after modifying a Bone2D node in any way, or closing and reopening the scene.

Color **editors/2d/bone_selected_color**

The color to use for selected bones in the 2D skeleton editor. See also editors/2d/bone_outline_color.

---

float **editors/2d/bone_width**

The bone width in the 2D skeleton editor (in pixels). See also editors/2d/bone_outline_size.

Note: Changes to this value only apply after modifying a Bone2D node in any way, or closing and reopening the scene.

Color **editors/2d/grid_color**

The grid color to use in the 2D editor.

Color **editors/2d/guides_color**

The guides color to use in the 2D editor. Guides can be created by dragging the mouse cursor from the rulers.

Color **editors/2d/smart_snapping_line_color**

The color to use when drawing smart snapping lines in the 2D editor. The smart snapping lines will automatically display when moving 2D nodes if smart snapping is enabled in the Snapping Options menu at the top of the 2D editor viewport.

---

bool **editors/2d/use_integer_zoom_by_default**

If `true`, the 2D editor will snap to integer zoom values while not holding the `Alt` key and powers of two while holding it. If `false`, this behavior is swapped.

Color **editors/2d/viewport_border_color** 🔗

The color of the viewport border in the 2D editor. This border represents the viewport's size at the base resolution defined in the Project Settings. Objects placed outside this border will not be visible unless a Camera2D node is used, or unless the window is resized and the stretch mode is set to `disabled`.

---

float **editors/3d/default_fov** 🔗

The default camera vertical field of view to use in the 3D editor (in degrees). The camera field of view can be adjusted on a per-scene basis using the **View** menu at the top of the 3D editor. If a scene had its camera field of view adjusted using the **View** menu, this setting is ignored in the scene in question. This setting is also ignored while a Camera3D node is being previewed in the editor.

Note: The editor camera always uses the **Keep Height** aspect mode.

---

float **editors/3d/default_z_far** 🔗

The default camera far clip distance to use in the 3D editor (in degrees). Higher values make it possible to view objects placed further away from the camera, at the cost of lower precision in the depth buffer (which can result in visible Z-fighting in the distance). The camera far clip distance can be adjusted on a per-scene basis using the **View** menu at the top of the 3D editor. If a scene had its camera far clip distance adjusted using the **View** menu, this setting is ignored in the scene in question. This setting is also ignored while a Camera3D node is being previewed in the editor.

---

float **editors/3d/default_z_near** 🔗

The default camera near clip distance to use in the 3D editor (in degrees). Lower values make it possible to view objects placed closer to the camera, at the cost of lower precision in the depth buffer (which can result in visible Z-fighting in the distance). The camera near clip distance can be adjusted on a per-scene basis using the **View** menu at the top of the 3D editor. If a scene had its camera near clip distance adjusted using the **View** menu, this setting is ignored in the scene in question. This setting is also ignored while a Camera3D node is being previewed in the editor.

---

int **editors/3d/freelook/freelook_activation_modifier** 🔗

The modifier key to use to enable freelook in the 3D editor (on top of pressing the right mouse button).

Note: Regardless of this setting, the freelook toggle keyboard shortcut (`Shift + F` by default) is always available.

Note: On certain window managers on Linux, the `Alt` key will be intercepted by the window manager when clicking a mouse button at the same time. This means Godot will not see the modifier key as being pressed.

---

float **editors/3d/freelook/freelook_base_speed** 🔗

The base 3D freelook speed in units per second. This can be adjusted by using the mouse wheel while in freelook mode, or by holding down the "fast" or "slow" modifier keys (`Shift` and `Alt` by default, respectively).

float **editors/3d/freelook/freelook_inertia** 🔗

The inertia of the 3D freelook camera. Higher values make the camera start and stop slower, which looks smoother but adds latency.

int **editors/3d/freelook/freelook_navigation_scheme** 🔗

The navigation scheme to use when freelook is enabled in the 3D editor. Some of the navigation schemes below may be more convenient when designing specific levels in the 3D editor.

- **Default:** The "Freelook Forward", "Freelook Backward", "Freelook Up" and "Freelook Down" keys will move relative to the camera, taking its pitch angle into account for the movement.
- **Partially Axis-Locked:** The "Freelook Forward" and "Freelook Backward" keys will move relative to the camera, taking its pitch angle into account for the movement. The "Freelook Up" and "Freelook Down" keys will move in an "absolute" manner, *not* taking the camera's pitch angle into account for the movement.
- **Fully Axis-Locked:** The "Freelook Forward", "Freelook Backward", "Freelook Up" and "Freelook Down" keys will move in an "absolute" manner, *not* taking the camera's pitch angle into account for the movement.

See also editors/3d/navigation/navigation_scheme.

float **editors/3d/freelook/freelook_sensitivity** 🔗

The mouse sensitivity to use while freelook mode is active in the 3D editor. See also editors/3d/navigation_feel/orbit_sensitivity.

bool **editors/3d/freelook/freelook_speed_zoom_link** 🔗

If `true`, freelook speed is linked to the zoom value used in the camera orbit mode in the 3D editor.

float **editors/3d/grid_division_level_bias** 🔗

The grid division bias to use in the 3D editor. Negative values will cause small grid divisions to appear earlier, whereas positive values will cause small grid divisions to appear later.

int **editors/3d/grid_division_level_max** 🔗

The largest grid division to use in the 3D editor. Together with editors/3d/primary_grid_steps, this determines how large the grid divisions can be. The grid divisions will not be able to get larger than `primary_grid_steps ^ grid_division_level_max` units. By default, when editors/3d/primary_grid_steps is `8`, this means grid divisions cannot get larger than `64` units each (so primary grid lines are `512` units apart), no matter how far away the camera is from the grid.

int **editors/3d/grid_division_level_min** 🔗

The smallest grid division to use in the 3D editor. Together with editors/3d/primary_grid_steps, this determines how small the grid divisions can be. The grid divisions will not be able to get smaller than `primary_grid_steps ^ grid_division_level_min` units. By default, this means grid divisions cannot get smaller than 1 unit each, no matter how close the camera is from the grid.

int **editors/3d/grid_size** 🔗

The grid size in units. Higher values prevent the grid from appearing "cut off" at certain angles, but make the grid more demanding to render. Depending on the camera's position, the grid may not be fully visible since a shader is used to fade it progressively.

bool **editors/3d/grid_xy_plane** 🔗

If `true`, render the grid on an XY plane. This can be useful for 3D side-scrolling games.

bool **editors/3d/grid_xz_plane** 🔗

If `true`, render the grid on an XZ plane.

bool **editors/3d/grid_yz_plane** 🔗

If `true`, render the grid on a YZ plane. This can be useful for 3D side-scrolling games.

bool **editors/3d/navigation/emulate_3_button_mouse** 🔗

If `true`, enables 3-button mouse emulation mode. This is useful on laptops when using a trackpad.

When 3-button mouse emulation mode is enabled, the pan, zoom and orbit modifiers can always be used in the 3D editor viewport, even when not holding down any mouse button.

Note: No matter the orbit modifier configured in editors/3d/navigation/orbit_modifier, `Alt` will always remain usable for orbiting in this mode to improve usability with graphics tablets.

bool **editors/3d/navigation/emulate_numpad** 🔗

If `true`, allows using the top row `0-9` keys to function as their equivalent numpad keys for 3D editor navigation. This should be enabled on keyboards that have no numeric keypad available.

bool **editors/3d/navigation/invert_x_axis** 🔗

If `true`, invert the horizontal mouse axis when panning or orbiting in the 3D editor. This setting does *not* apply to freelook mode.

bool **editors/3d/navigation/invert_y_axis** 🔗

If `true`, invert the vertical mouse axis when panning, orbiting, or using freelook mode in the 3D editor.

---

int **editors/3d/navigation/navigation_scheme**

The navigation scheme to use in the 3D editor. Changing this setting will affect the mouse buttons that must be held down to perform certain operations in the 3D editor viewport.

- **Godot** Middle mouse button to orbit, `Shift + Middle mouse button` to pan. `Mouse wheel` to zoom.
- **Maya:** `Alt + Left mouse button` to orbit. `Middle mouse button` to pan, `Shift + Middle mouse button` to pan 10 times faster. `Mouse wheel` to zoom.
- **Modo:** `Alt + Left mouse button` to orbit. `Alt + Shift + Left mouse button` to pan. `Ctrl + Alt + Left mouse button` to zoom.

See also editors/3d/freelook/freelook_navigation_scheme.

Note: On certain window managers on Linux, the `Alt` key will be intercepted by the window manager when clicking a mouse button at the same time. This means Godot will not see the modifier key as being pressed.

---

int **editors/3d/navigation/orbit_modifier**

The modifier key that must be held to orbit in the 3D editor.

Note: If editors/3d/navigation/emulate_3_button_mouse is `true`, `Alt` will always remain usable for orbiting to improve usability with graphics tablets.

Note: On certain window managers on Linux, the `Alt` key will be intercepted by the window manager when clicking a mouse button at the same time. This means Godot will not see the modifier key as being pressed.

---

int **editors/3d/navigation/pan_modifier**

The modifier key that must be held to pan in the 3D editor.

Note: On certain window managers on Linux, the `Alt` key will be intercepted by the window manager when clicking a mouse button at the same time. This means Godot will not see the modifier key as being pressed.

---

bool **editors/3d/navigation/warped_mouse_panning**

If `true`, warps the mouse around the 3D viewport while panning in the 3D editor. This makes it possible to pan over a large area without having to exit panning and adjust the mouse cursor.

---

int **editors/3d/navigation/zoom_modifier**

The modifier key that must be held to zoom in the 3D editor.

Note: On certain window managers on Linux, the `Alt` key will be intercepted by the window manager when clicking a mouse button at the same time. This means Godot will not see the modifier key as being pressed.

int **editors/3d/navigation/zoom_style**

The mouse cursor movement direction to use when zooming by moving the mouse. This does not affect zooming with the mouse wheel.

float **editors/3d/navigation_feel/orbit_inertia**

The inertia to use when orbiting in the 3D editor. Higher values make the camera start and stop slower, which looks smoother but adds latency.

float **editors/3d/navigation_feel/orbit_sensitivity**

The mouse sensitivity to use when orbiting in the 3D editor. See also editors/3d/freelook/freelook_sensitivity.

float **editors/3d/navigation_feel/translation_inertia**

The inertia to use when panning in the 3D editor. Higher values make the camera start and stop slower, which looks smoother but adds latency.

float **editors/3d/navigation_feel/zoom_inertia**

The inertia to use when zooming in the 3D editor. Higher values make the camera start and stop slower, which looks smoother but adds latency.

Color **editors/3d/primary_grid_color**

The color to use for the primary 3D grid. The color's alpha channel affects the grid's opacity.

int **editors/3d/primary_grid_steps**

If set above 0, where a primary grid line should be drawn. By default, primary lines are configured to be more visible than secondary lines. This helps with measurements in the 3D editor. See also editors/3d/primary_grid_color and editors/3d/secondary_grid_color.

Color **editors/3d/secondary_grid_color**

The color to use for the secondary 3D grid. This is generally a less visible color than editors/3d/primary_grid_color. The color's alpha channel affects the grid's opacity.

Color **editors/3d/selection_box_color**

The color to use for the selection box that surrounds selected nodes in the 3D editor viewport. The color's alpha channel influences the selection box's opacity.

Color **editors/3d_gizmos/gizmo_colors/aabb** 🔗

The color to use for the AABB gizmo that displays the GeometryInstance3D's custom AABB.

Color **editors/3d_gizmos/gizmo_colors/instantiated** 🔗

The color override to use for 3D editor gizmos if the Node3D in question is part of an instantiated scene file (from the perspective of the current scene).

Color **editors/3d_gizmos/gizmo_colors/joint** 🔗

The 3D editor gizmo color for Joint3Ds and PhysicalBone3Ds.

bool **editors/animation/autorename_animation_tracks** 🔗

If `true`, automatically updates animation tracks' target paths when renaming or reparenting nodes in the Scene tree dock.

bool **editors/animation/confirm_insert_track** 🔗

If `true`, display a confirmation dialog when adding a new track to an animation by pressing the "key" icon next to a property. Holding Shift will bypass the dialog.

If `false`, the behavior is reversed, i.e. the dialog only appears when Shift is held.

bool **editors/animation/default_create_bezier_tracks** 🔗

If `true`, create a Bezier track instead of a standard track when pressing the "key" icon next to a property. Bezier tracks provide more control over animation curves, but are more difficult to adjust quickly.

bool **editors/animation/default_create_reset_tracks** 🔗

If `true`, create a `RESET` track when creating a new animation track. This track can be used to restore the animation to a "default" state.

Color **editors/animation/onion_layers_future_color** 🔗

The modulate color to use for "future" frames displayed in the animation editor's onion skinning feature.

Color **editors/animation/onion_layers_past_color** 🔗

The modulate color to use for "past" frames displayed in the animation editor's onion skinning feature.

float **editors/grid_map/pick_distance** 🔗

The maximum distance at which tiles can be placed on a GridMap, relative to the camera position (in 3D units).

---

int **editors/panning/2d_editor_pan_speed** 🔗

The panning speed when using the mouse wheel or touchscreen events in the 2D editor. This setting does not apply to panning by holding down the middle or right mouse buttons.

---

int **editors/panning/2d_editor_panning_scheme** 🔗

Controls whether the mouse wheel scroll zooms or pans in the 2D editor. See also editors/panning/sub_editors_panning_scheme and editors/panning/animation_editors_panning_scheme.

---

int **editors/panning/animation_editors_panning_scheme** 🔗

Controls whether the mouse wheel scroll zooms or pans in the animation track and Bezier editors. See also editors/panning/2d_editor_panning_scheme and editors/panning/sub_editors_panning_scheme (which controls the animation blend tree editor's pan behavior).

---

bool **editors/panning/simple_panning** 🔗

If `true`, allows panning by holding down `Space` in the 2D editor viewport (in addition to panning with the middle or right mouse buttons). If `false`, the left mouse button must be held down while holding down `Space` to pan in the 2D editor viewport.

---

int **editors/panning/sub_editors_panning_scheme** 🔗

Controls whether the mouse wheel scroll zooms or pans in subeditors. The list of affected subeditors is: animation blend tree editor, Polygon2D editor, tileset editor, texture region editor and visual shader editor. See also editors/panning/2d_editor_panning_scheme and editors/panning/animation_editors_panning_scheme.

---

bool **editors/panning/warped_mouse_panning** 🔗

If `true`, warps the mouse around the 2D viewport while panning in the 2D editor. This makes it possible to pan over a large area without having to exit panning and adjust the mouse cursor.

---

float **editors/polygon_editor/auto_bake_delay** 🔗

The delay in seconds until more complex and performance costly polygon editors commit their outlines, e.g. the 2D navigation polygon editor rebakes the navigation mesh polygons. A negative value stops the auto bake.

---

int **editors/polygon_editor/point_grab_radius** 🔗

The radius in which points can be selected in the Polygon2D and CollisionPolygon2D editors (in pixels). Higher values make it easier to select points quickly, but can make it more difficult to select the expected point when several points are located close to each other.

---

bool **editors/polygon_editor/show_previous_outline**

If `true`, displays the polygon's previous shape in the 2D polygon editors with an opaque gray outline. This outline is displayed while dragging a point until the left mouse button is released.

---

bool **editors/shader_editor/behavior/files/restore_shaders_on_load**

If `true`, reopens shader files that were open in the shader editor when the project was last closed.

---

bool **editors/tiles_editor/display_grid**

If `true`, displays a grid while the TileMap editor is active. See also editors/tiles_editor/grid_color.

Color **editors/tiles_editor/grid_color**

The color to use for the TileMap editor's grid.

Note: Only effective if editors/tiles_editor/display_grid is `true`.

---

bool **editors/tiles_editor/highlight_selected_layer**

Highlight the currently selected TileMapLayer by dimming the other ones in the scene.

Color **editors/visual_editors/category_colors/color_color**

The color of a graph node's header when it belongs to the "Color" category.

Color **editors/visual_editors/category_colors/conditional_color**

The color of a graph node's header when it belongs to the "Conditional" category.

Color **editors/visual_editors/category_colors/input_color**

The color of a graph node's header when it belongs to the "Input" category.

Color **editors/visual_editors/category_colors/output_color**

The color of a graph node's header when it belongs to the "Output" category.

Color **editors/visual_editors/category_colors/particle_color**

The color of a graph node's header when it belongs to the "Particle" category.

Color **editors/visual_editors/category_colors/scalar_color** 🔗

The color of a graph node's header when it belongs to the "Scalar" category.

Color **editors/visual_editors/category_colors/special_color** 🔗

The color of a graph node's header when it belongs to the "Special" category.

Color **editors/visual_editors/category_colors/textures_color** 🔗

The color of a graph node's header when it belongs to the "Textures" category.

Color **editors/visual_editors/category_colors/transform_color** 🔗

The color of a graph node's header when it belongs to the "Transform" category.

Color **editors/visual_editors/category_colors/utility_color** 🔗

The color of a graph node's header when it belongs to the "Utility" category.

Color **editors/visual_editors/category_colors/vector_color** 🔗

The color of a graph node's header when it belongs to the "Vector" category.

String **editors/visual_editors/color_theme** 🔗

The color theme to use in the visual shader editor.

Color **editors/visual_editors/connection_colors/boolean_color** 🔗

The color of a port/connection of boolean type.

Color **editors/visual_editors/connection_colors/sampler_color** 🔗

The color of a port/connection of sampler type.

Color **editors/visual_editors/connection_colors/scalar_color** 🔗

The color of a port/connection of scalar type (float, int, unsigned int).

Color **editors/visual_editors/connection_colors/transform_color** 🔗

The color of a port/connection of transform type.

Color **editors/visual_editors/connection_colors/vector2_color** 🔗

The color of a port/connection of Vector2 type.

Color **editors/visual_editors/connection_colors/vector3_color** 🔗

The color of a port/connection of Vector3 type.

Color **editors/visual_editors/connection_colors/vector4_color** 🔗

The color of a port/connection of Vector4 type.

---

int **editors/visual_editors/grid_pattern** 🔗

The pattern used for the background grid.

---

float **editors/visual_editors/lines_curvature** 🔗

The curvature to use for connection lines in the visual shader editor. Higher values will make connection lines appear more curved, with values above `0.5` resulting in more "angular" turns in the middle of connection lines.

---

float **editors/visual_editors/minimap_opacity** 🔗

The opacity of the minimap displayed in the bottom-right corner of the visual shader editor.

---

int **editors/visual_editors/visual_shader/port_preview_size** 🔗

The size to use for port previews in the visual shader uniforms (toggled by clicking the "eye" icon next to an output). The value is defined in pixels at 100% zoom, and will scale with zoom automatically.

String **filesystem/directories/autoscan_project_path** 🔗

The folder where projects should be scanned for (recursively), in a way similar to the project manager's **Scan** button. This can be set to the same value as filesystem/directories/default_project_path for convenience.

Note: Setting this path to a folder with very large amounts of files/folders can slow down the project manager startup significantly. To keep the project manager quick to start up, it is recommended to set this value to a folder as "specific" as possible.

String **filesystem/directories/default_project_path** 🔗

The folder where new projects should be created by default when clicking the project manager's **New Project** button. This can be set to the same value as filesystem/directories/autoscan_project_path for convenience.

String **filesystem/external_programs/3d_model_editor** 🔗

The program that opens 3D model scene files when clicking "Open in External Program" option in Filesystem Dock. If not specified, the file will be opened in the system's default program.

String **filesystem/external_programs/audio_editor** 🔗

The program that opens audio files when clicking "Open in External Program" option in Filesystem Dock. If not specified, the file will be opened in the system's default program.

String **filesystem/external_programs/raster_image_editor** 🔗

The program that opens raster image files when clicking "Open in External Program" option in Filesystem Dock. If not specified, the file will be opened in the system's default program.

String **filesystem/external_programs/terminal_emulator** 🔗

The terminal emulator program to use when using **Open in Terminal** context menu action in the FileSystem dock. You can enter an absolute path to a program binary, or a path to a program that is present in the `PATH` environment variable.

If left empty, Godot will use the default terminal emulator for the system:

- **Windows:** PowerShell
- **macOS:** Terminal.app
- **Linux:** The first terminal found on the system in this order: gnome-terminal, konsole, xfce4-terminal, lxterminal, kitty, alacritty, urxvt, xterm.

To use Command Prompt (cmd) instead of PowerShell on Windows, enter `cmd` in this field and the correct flags will automatically be used.

On macOS, make sure to point to the actual program binary located within the `Programs/MacOS` folder of the .app bundle, rather than the .app bundle directory.

If specifying a custom terminal emulator, you may need to override filesystem/external_programs/terminal_emulator_flags so it opens in the correct folder.

String **filesystem/external_programs/terminal_emulator_flags** 🔗

The command-line arguments to pass to the terminal emulator that is run when using **Open in Terminal** context menu action in the FileSystem dock. See also filesystem/external_programs/terminal_emulator.

If left empty, the default flags are `{directory}`, which is replaced by the absolute path to the directory that is being opened in the terminal.

Note: If the terminal emulator is set to PowerShell, cmd, or Konsole, Godot will automatically prepend arguments to this list, as these terminals require nonstandard arguments to open in the correct folder.

String **filesystem/external_programs/vector_image_editor** 🔗

The program that opens vector image files when clicking "Open in External Program" option in Filesystem Dock. If not specified, the file will be opened in the system's default program.

---

int **filesystem/file_dialog/display_mode**

The display mode to use in the editor's file dialogs.

- **Thumbnails** takes more space, but displays dynamic resource thumbnails, making resources easier to preview without having to open them.
- **List** is more compact but doesn't display dynamic resource thumbnails. Instead, it displays static icons based on the file extension.

---

bool **filesystem/file_dialog/show_hidden_files**

If `true`, display hidden files in the editor's file dialogs. Files that have names starting with `.` are considered hidden (e.g. `.hidden_file`).

---

int **filesystem/file_dialog/thumbnail_size**

The thumbnail size to use in the editor's file dialogs (in pixels). See also docks/filesystem/thumbnail_size.

String **filesystem/import/blender/blender_path**

The path to the directory containing the Blender executable used for converting the Blender 3D scene files `.blend` to glTF 2.0 format during import. Blender 3.0 or later is required.

To enable this feature for your specific project, use ProjectSettings.filesystem/import/blender/enabled.

---

int **filesystem/import/blender/rpc_port**

The port number used for Remote Procedure Call (RPC) communication with Godot's created process of the blender executable.

Setting this to 0 effectively disables communication with Godot and the blender process, making performance slower.

---

float **filesystem/import/blender/rpc_server_uptime**

The maximum idle uptime (in seconds) of the Blender process.

This prevents Godot from having to create a new process for each import within the given seconds.

String **filesystem/import/fbx/fbx2gltf_path**

The path to the FBX2glTF executable used for converting Autodesk FBX 3D scene files `.fbx` to glTF 2.0 format during import.

To enable this feature for your specific project, use ProjectSettings.filesystem/import/fbx2gltf/enabled.

---

bool **filesystem/on_save/compress_binary_resources**

If `true`, uses lossless compression for binary resources.

---

bool **filesystem/on_save/safe_save_on_backup_then_rename** 🔗

If `true`, when saving a file, the editor will rename the old file to a different name, save a new file, then only remove the old file once the new file has been saved. This makes loss of data less likely to happen if the editor or operating system exits unexpectedly while saving (e.g. due to a crash or power outage).

Note: On Windows, this feature can interact negatively with certain antivirus programs. In this case, you may have to set this to `false` to prevent file locking issues.

String **filesystem/tools/oidn/oidn_denoise_path** 🔗

The path to the directory containing the Open Image Denoise (OIDN) executable, used optionally for denoising lightmaps. It can be downloaded from openimagedenoise.org [https://www.openimagedenoise.org/downloads.html].

To enable this feature for your specific project, use ProjectSettings.rendering/lightmapping/denoising/denoiser.

---

bool **input/buffering/agile_event_flushing** 🔗

If `true`, input events will be flushed just before every idle and physics frame.

If `false`, these events will be flushed only once per process frame, between iterations of the engine.

Enabling this setting can greatly improve input responsiveness, especially in devices that struggle to run at the project's intended frame rate.

---

bool **input/buffering/use_accumulated_input** 🔗

If `true`, similar input events sent by the operating system are accumulated. When input accumulation is enabled, all input events generated during a frame will be merged and emitted when the frame is done rendering. Therefore, this limits the number of input method calls per second to the rendering FPS.

Input accumulation can be disabled to get slightly more precise/reactive input at the cost of increased CPU usage.

Note: Input accumulation is *enabled* by default.

---

int **interface/editor/accept_dialog_cancel_ok_buttons** 🔗

How to position the Cancel and OK buttons in the editor's AcceptDialogs. Different platforms have different standard behaviors for this, which can be overridden using this setting. This is useful if you use Godot both on Windows and macOS/Linux and your Godot muscle memory is stronger than your OS specific one.

- **Auto** follows the platform convention: Cancel first on macOS and Linux, OK first on Windows.
- **Cancel First** forces the ordering Cancel/OK.
- **OK First** forces the ordering OK/Cancel.

---

bool **interface/editor/automatically_open_screenshots** 🔗

If `true`, automatically opens screenshots with the default program associated to `.png` files after a screenshot is taken using the **Editor > Take Screenshot** action.

String **interface/editor/code_font** 🔗

The font to use for the script editor. Must be a resource of a Font type such as a `.ttf` or `.otf` font file.

---

int **interface/editor/code_font_contextual_ligatures** 🔗

The font ligatures to enable for the currently configured code font. Not all fonts include support for ligatures.

Note: The default editor code font (JetBrains Mono [https://www.jetbrains.com/lp/mono/]) has contextual ligatures in its font file.

String **interface/editor/code_font_custom_opentype_features** 🔗

List of custom OpenType features to use, if supported by the currently configured code font. Not all fonts include support for custom OpenType features. The string should follow the OpenType specification.

Note: The default editor code font (JetBrains Mono [https://www.jetbrains.com/lp/mono/]) has custom OpenType features in its font file, but there is no documented list yet.

String **interface/editor/code_font_custom_variations** 🔗

List of alternative characters to use, if supported by the currently configured code font. Not all fonts include support for custom variations. The string should follow the OpenType specification.

Note: The default editor code font (JetBrains Mono [https://www.jetbrains.com/lp/mono/]) has alternate characters in its font file, but there is no documented list yet.

---

int **interface/editor/code_font_size** 🔗

The size of the font in the script editor. This setting does not impact the font size of the Output panel (see run/output/font_size).

---

float **interface/editor/custom_display_scale** 🔗

The custom editor scale factor to use. This can be used for displays with very high DPI where a scale factor of 200% is not sufficient.

Note: Only effective if interface/editor/display_scale is set to **Custom**.

---

int **interface/editor/display_scale** 🔗

The display scale factor to use for the editor interface. Higher values are more suited to hiDPI/Retina displays.

If set to **Auto**, the editor scale is automatically determined based on the screen resolution and reported display DPI. This heuristic is not always ideal, which means you can get better results by setting the editor scale manually.

If set to **Custom**, the scaling value in interface/editor/custom_display_scale will be used.

---

int **interface/editor/dock_tab_style** 🔗

Tab style of editor docks.

String **interface/editor/editor_language** 🔗

The language to use for the editor interface.

Translations are provided by the community. If you spot a mistake, contribute to editor translations on Weblate!

---

int **interface/editor/editor_screen** 🔗

The preferred monitor to display the editor.

---

bool **interface/editor/expand_to_title** 🔗

Expanding main editor window content to the title, if supported by DisplayServer. See DisplayServer.WINDOW_FLAG_EXTEND_TO_TITLE.

Specific to the macOS platform.

---

int **interface/editor/font_antialiasing** 🔗

FreeType's font anti-aliasing mode used to render the editor fonts. Most fonts are not designed to look good with anti-aliasing disabled, so it's recommended to leave this enabled unless you're using a pixel art font.

---

bool **interface/editor/font_disable_embedded_bitmaps** 🔗

If set to `true`, embedded font bitmap loading is disabled (bitmap-only and color fonts ignore this property).

---

int **interface/editor/font_hinting** 🔗

The font hinting mode to use for the editor fonts. FreeType supports the following font hinting modes:

- **None:** Don't use font hinting when rasterizing the font. This results in a smooth font, but it can look blurry.

- **Light:** Use hinting on the X axis only. This is a compromise between font sharpness and smoothness.
- **Normal:** Use hinting on both X and Y axes. This results in a sharp font, but it doesn't look very smooth.

If set to **Auto**, the font hinting mode will be set to match the current operating system in use. This means the **Light** hinting mode will be used on Windows and Linux, and the **None** hinting mode will be used on macOS.

---

int **interface/editor/font_subpixel_positioning** 🔗

The subpixel positioning mode to use when rendering editor font glyphs. This affects both the main and code fonts. **Disabled** is the fastest to render and uses the least memory. **Auto** only uses subpixel positioning for small font sizes (where the benefit is the most noticeable). **One Half of a Pixel** and **One Quarter of a Pixel** force the same subpixel positioning mode for all editor fonts, regardless of their size (with **One Quarter of a Pixel** being the highest-quality option).

---

bool **interface/editor/import_resources_when_unfocused** 🔗

If `true`, (re)imports resources even if the editor window is unfocused or minimized. If `false`, resources are only (re)imported when the editor window is focused. This can be set to `true` to speed up iteration by starting the import process earlier when saving files in the project folder. This also allows getting visual feedback on changes without having to click the editor window, which is useful with multi-monitor setups. The downside of setting this to `true` is that it increases idle CPU usage and may steal CPU time from other applications when importing resources.

---

bool **interface/editor/localize_settings** 🔗

If `true`, setting names in the editor are localized when possible.

Note: This setting affects most EditorInspectors in the editor UI, primarily Project Settings and Editor Settings. To control names displayed in the Inspector dock, use interface/inspector/default_property_name_style instead.

---

int **interface/editor/low_processor_mode_sleep_usec** 🔗

The amount of sleeping between frames when the low-processor usage mode is enabled (in microseconds). Higher values will result in lower CPU/GPU usage, which can improve battery life on laptops. However, higher values will result in a less responsive editor. The default value is set to allow for maximum smoothness on monitors up to 144 Hz. See also interface/editor/unfocused_low_processor_mode_sleep_usec.

Note: This setting is ignored if interface/editor/update_continuously is `true`, as enabling that setting disables low-processor mode.

String **interface/editor/main_font** 🔗

The font to use for the editor interface. Must be a resource of a Font type such as a `.ttf` or `.otf` font file.

String **interface/editor/main_font_bold** 🔗

The font to use for bold text in the editor interface. Must be a resource of a Font type such as a `.ttf` or `.otf` font file.

---

int **interface/editor/main_font_size** 🔗

The size of the font in the editor interface.

---

bool **interface/editor/mouse_extra_buttons_navigate_history** 🔗

If `true`, the mouse's additional side buttons will be usable to navigate in the script editor's file history. Set this to `false` if you're using the side buttons for other purposes (such as a push-to-talk button in a VoIP program).

---

int **interface/editor/project_manager_screen** 🔗

The preferred monitor to display the project manager.

---

bool **interface/editor/save_each_scene_on_quit** 🔗

If `false`, the editor will save all scenes when confirming the **Save** action when quitting the editor or quitting to the project list. If `true`, the editor will ask to save each scene individually.

---

bool **interface/editor/save_on_focus_loss** 🔗

If `true`, scenes and scripts are saved when the editor loses focus. Depending on the work flow, this behavior can be less intrusive than text_editor/behavior/files/autosave_interval_secs or remembering to save manually.

---

bool **interface/editor/separate_distraction_mode** 🔗

If `true`, the editor's Script tab will have a separate distraction mode setting from the 2D/3D/AssetLib tabs. If `false`, the distraction-free mode toggle is shared between all tabs.

---

int **interface/editor/show_internal_errors_in_toast_notifications** 🔗

If enabled, displays internal engine errors in toast notifications (toggleable by clicking the "bell" icon at the bottom of the editor). No matter the value of this setting, non-internal engine errors will always be visible in toast notifications.

The default **Auto** value will only enable this if the editor was compiled with the `dev_build=yes` SCons option (the default is `dev_build=no`).

---

int **interface/editor/show_update_spinner** 🔗

If enabled, displays an icon in the top-right corner of the editor that spins when the editor redraws a frame. This can be used to diagnose situations where the engine is constantly redrawing, which should be avoided as this increases CPU and GPU utilization for no good reason. To further troubleshoot these situations, start the editor with the `--debug-canvas-item-redraw` command line argument.

Consider enabling this if you are developing editor plugins to ensure they only make the editor redraw when required.

The default **Auto** value will only enable this if the editor was compiled with the `dev_build=yes` SCons option (the default is `dev_build=no`).

Note: If interface/editor/update_continuously is `true`, the spinner icon displays in red.

Note: If the editor was started with the `--debug-canvas-item-redraw` command line argument, the update spinner will *never* display regardless of this setting's value. This is to avoid confusion with what would cause redrawing in real world scenarios.

---

bool **interface/editor/single_window_mode** 🔗

If `true`, embed modal windows such as docks inside the main editor window. When single-window mode is enabled, tooltips will also be embedded inside the main editor window, which means they can't be displayed outside of the editor window. Single-window mode can be faster as it does not need to create a separate window for every popup and tooltip, which can be a slow operation depending on the operating system and rendering method in use.

This is equivalent to ProjectSettings.display/window/subwindows/embed_subwindows in the running project, except the setting's value is inverted.

Note: To query whether the editor can use multiple windows in an editor plugin, use EditorInterface.is_multi_window_enabled instead of querying the value of this editor setting.

---

int **interface/editor/ui_layout_direction** 🔗

Editor UI default layout direction.

---

int **interface/editor/unfocused_low_processor_mode_sleep_usec** 🔗

When the editor window is unfocused, the amount of sleeping between frames when the low-processor usage mode is enabled (in microseconds). Higher values will result in lower CPU/GPU usage, which can improve battery life on laptops (in addition to improving the running project's performance if the editor has to redraw continuously). However, higher values will result in a less responsive editor. The default value is set to limit the editor to 20 FPS when the editor window is unfocused. See also interface/editor/low_processor_mode_sleep_usec.

Note: This setting is ignored if interface/editor/update_continuously is `true`, as enabling that setting disables low-processor mode.

---

bool **interface/editor/update_continuously** 🔗

If `true`, redraws the editor every frame even if nothing has changed on screen. When this setting is enabled, the update spinner displays in red (see interface/editor/show_update_spinner).

Warning: This greatly increases CPU and GPU utilization, leading to increased power usage. This should only be enabled for troubleshooting purposes.

---

bool **interface/editor/use_embedded_menu** 🔗

If `true`, editor main menu is using embedded MenuBar instead of system global menu.

Specific to the macOS platform.

---

bool **interface/editor/use_native_file_dialogs** 🔗

If `true`, editor UI uses OS native file/directory selection dialogs.

---

int **interface/editor/vsync_mode** 🔗

Sets the V-Sync mode for the editor. Does not affect the project when run from the editor (this is controlled by ProjectSettings.display/window/vsync/vsync_mode).

Depending on the platform and used renderer, the engine will fall back to **Enabled** if the desired mode is not supported.

Note: V-Sync modes other than **Enabled** are only supported in the Forward+ and Mobile rendering methods, not Compatibility.

---

bool **interface/inspector/auto_unfold_foreign_scenes** 🔗

If `true`, automatically expands property groups in the Inspector dock when opening a scene that hasn't been opened previously. If `false`, all groups remain collapsed by default.

---

int **interface/inspector/default_color_picker_mode** 🔗

The default color picker mode to use when opening ColorPickers in the editor. This mode can be temporarily adjusted on the color picker itself.

---

int **interface/inspector/default_color_picker_shape** 🔗

The default color picker shape to use when opening ColorPickers in the editor. This shape can be temporarily adjusted on the color picker itself.

---

float **interface/inspector/default_float_step** 🔗

The floating-point precision to use for properties that don't define an explicit precision step. Lower values allow entering more precise values.

---

int **interface/inspector/default_property_name_style** 🔗

The default property name style to display in the Inspector dock. This style can be temporarily adjusted in the Inspector dock's menu.

- **Raw:** Displays properties in `snake_case`.
- **Capitalized:** Displays properties capitalized.
- **Localized:** Displays the localized string for the current editor language if a translation is available for the given property. If no translation is available, falls back to **Capitalized**.

Note: To display translated setting names in Project Settings and Editor Settings, use interface/editor/localize_settings instead.

---

bool **interface/inspector/delimitate_all_container_and_resources**

If `true`, add a margin around Array, Dictionary, and Resource Editors that are not already colored.

Note: If interface/inspector/nested_color_mode is set to **Containers & Resources** this parameter will have no effect since those editors will already be colored

---

bool **interface/inspector/disable_folding**

If `true`, forces all property groups to be expanded in the Inspector dock and prevents collapsing them.

---

float **interface/inspector/float_drag_speed**

Base speed for increasing/decreasing float values by dragging them in the inspector.

---

bool **interface/inspector/horizontal_vector2_editing**

If `true`, Vector2 and Vector2i properties are shown on a single line in the inspector instead of two lines. This is overall more compact, but it can be harder to view and edit large values without expanding the inspector horizontally.

---

bool **interface/inspector/horizontal_vector_types_editing**

If `true`, Vector3, Vector3i, Vector4, Vector4i, Rect2, Rect2i, Plane, and Quaternion properties are shown on a single line in the inspector instead of multiple lines. This is overall more compact, but it can be harder to view and edit large values without expanding the inspector horizontally.

---

int **interface/inspector/max_array_dictionary_items_per_page**

The number of Array or Dictionary items to display on each "page" in the inspector. Higher values allow viewing more values per page, but take more time to load. This increased load time is noticeable when selecting nodes that have array or dictionary properties in the editor.

---

int **interface/inspector/nested_color_mode**

Control which property editors are colored when they are opened.

- **Containers & Resources:** Color all Array, Dictionary, and Resource Editors.
- **Resources:** Color all Resource Editors.
- **External Resources:** Color Resource Editors that edits an external resource.

---

bool **interface/inspector/open_resources_in_current_inspector** 🔗

If `true`, subresources can be edited in the current inspector view. If the resource type is defined in interface/inspector/resources_to_open_in_new_inspector or if this setting is `false`, attempting to edit a subresource always opens a new inspector view.

PackedStringArray **interface/inspector/resources_to_open_in_new_inspector** 🔗

List of resources that should always be opened in a new inspector view, even if interface/inspector/open_resources_in_current_inspector is `true`.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedStringArray for more details.

---

bool **interface/inspector/show_low_level_opentype_features** 🔗

If `true`, display OpenType features marked as `hidden` by the font file in the Font editor.

---

bool **interface/multi_window/enable** 🔗

If `true`, multiple window support in editor is enabled. The following panels can become dedicated windows (i.e. made floating): Docks, Script editor, and Shader editor.

Note: When interface/editor/single_window_mode is `true`, the multi window support is always disabled.

Note: To query whether the editor can use multiple windows in an editor plugin, use EditorInterface.is_multi_window_enabled instead of querying the value of this editor setting.

---

bool **interface/multi_window/maximize_window** 🔗

If `true`, when panels are made floating they will be maximized.

If `false`, when panels are made floating their position and size will match the ones when they are attached (excluding window border) to the editor window.

---

bool **interface/multi_window/restore_windows_on_load** 🔗

If `true`, the floating panel position, size, and screen will be saved on editor exit. On next launch the panels that were floating will be made floating in the saved positions, sizes and screens, if possible.

---

int **interface/scene_tabs/display_close_button** 🔗

Controls when the Close (X) button is displayed on scene tabs at the top of the editor.

---

int **interface/scene_tabs/maximum_width** 🔗

The maximum width of each scene tab at the top editor (in pixels).

---

bool **interface/scene_tabs/restore_scenes_on_load** 🔗

If `true`, when a project is loaded, restores scenes that were opened on the last editor session.

Note: With many opened scenes, the editor may take longer to become usable. If starting the editor quickly is necessary, consider setting this to `false`.

---

bool **interface/scene_tabs/show_script_button** 🔗

If `true`, show a button next to each scene tab that opens the scene's "dominant" script when clicked. The "dominant" script is the one that is at the highest level in the scene's hierarchy.

---

bool **interface/scene_tabs/show_thumbnail_on_hover** 🔗

If `true`, display an automatically-generated thumbnail when hovering scene tabs with the mouse. Scene thumbnails are generated when saving the scene.

Color **interface/theme/accent_color** 🔗

The color to use for "highlighted" user interface elements in the editor (pressed and hovered items).

---

int **interface/theme/additional_spacing** 🔗

The extra spacing to add to various GUI elements in the editor (in pixels). Increasing this value is useful to improve usability on touch screens, at the cost of reducing the amount of usable screen real estate.

See also interface/theme/spacing_preset.

Color **interface/theme/base_color** 🔗

The base color to use for user interface elements in the editor. Secondary colors (such as darker/lighter variants) are derived from this color.

---

int **interface/theme/base_spacing** 🔗

The base spacing used by various GUI elements in the editor (in pixels). See also interface/theme/spacing_preset.

---

int **interface/theme/border_size** 🔗

The border size to use for interface elements (in pixels).

---

float **interface/theme/contrast** 🔗

The contrast factor to use when deriving the editor theme's base color (see interface/theme/base_color). When using a positive values, the derived colors will be *darker* than the base color. This contrast factor can be set to a negative value, which will make the derived colors *brighter* than the base color. Negative contrast rates often look better for light themes.

---

int **interface/theme/corner_radius** 🔗

The corner radius to use for interface elements (in pixels). `0` is square.

String **interface/theme/custom_theme** 🔗

The custom theme resource to use for the editor. Must be a Godot theme resource in `.tres` or `.res` format.

---

bool **interface/theme/draw_extra_borders** 🔗

If `true`, draws additional borders around interactive UI elements in the editor. This is automatically enabled when using the **Black (OLED)** theme preset, as this theme preset uses a fully black background.

---

bool **interface/theme/follow_system_theme** 🔗

If `true`, the editor theme preset will attempt to automatically match the system theme.

---

int **interface/theme/icon_and_font_color** 🔗

The icon and font color scheme to use in the editor.

- **Auto** determines the color scheme to use automatically based on interface/theme/base_color.
- **Dark** makes fonts and icons dark (suitable for light themes). Icon colors are automatically converted by the editor following the set of rules defined in this file [https://github.com/godotengine/godot/blob/master/editor/themes/editor_theme_manager.cpp].
- **Light** makes fonts and icons light (suitable for dark themes).

---

float **interface/theme/icon_saturation** 🔗

The saturation to use for editor icons. Higher values result in more vibrant colors.

Note: The default editor icon saturation was increased by 30% in Godot 4.0 and later. To get Godot 3.x's icon saturation back, set interface/theme/icon_saturation to `0.77`.

String **interface/theme/preset** 🔗

The editor theme preset to use.

---

float **interface/theme/relationship_line_opacity** 🔗

The opacity to use when drawing relationship lines in the editor's Tree-based GUIs (such as the Scene tree dock).

String **interface/theme/spacing_preset** 🔗

The editor theme spacing preset to use. See also interface/theme/base_spacing and interface/theme/additional_spacing.

---

bool **interface/theme/use_system_accent_color** 🔗

If `true`, set accent color based on system settings.

Note: This setting is only effective on Windows and MacOS.

---

bool **interface/touchscreen/enable_long_press_as_right_click** 🔗

If `true`, long press on touchscreen is treated as right click.

Note: Defaults to `true` on touchscreen devices.

---

bool **interface/touchscreen/enable_pan_and_scale_gestures** 🔗

If `true`, enable two finger pan and scale gestures on touchscreen devices.

Note: Defaults to `true` on touchscreen devices.

---

bool **interface/touchscreen/increase_scrollbar_touch_area** 🔗

If `true`, increases the scrollbar touch area to improve usability on touchscreen devices.

Note: Defaults to `true` on touchscreen devices.

---

float **interface/touchscreen/scale_gizmo_handles** 🔗

Specify the multiplier to apply to the scale for the editor gizmo handles to improve usability on touchscreen devices.

Note: Defaults to `1` on non-touchscreen devices.

---

int **network/connection/engine_version_update_mode** 🔗

Specifies how the engine should check for updates.

- **Disable Update Checks** will block the engine from checking updates (see also network/connection/network_mode).
- **Check Newest Preview** (default for preview versions) will check for the newest available development snapshot.
- **Check Newest Stable** (default for stable versions) will check for the newest available stable version.
- **Check Newest Patch** will check for the latest available stable version, but only within the same minor version. E.g. if your version is `4.3.stable`, you will be notified about `4.3.1.stable`, but not

`4.4.stable`.

All update modes will ignore builds with different major versions (e.g. Godot 4 -> Godot 5).

int **network/connection/network_mode** 🔗

Determines whether online features are enabled in the editor, such as the Asset Library or update checks. Disabling these online features helps alleviate privacy concerns by preventing the editor from making HTTP requests to the Godot website or third-party platforms hosting assets from the Asset Library.

String **network/debug/remote_host** 🔗

The address to listen to when starting the remote debugger. This can be set to `0.0.0.0` to allow external clients to connect to the remote debugger (instead of restricting the remote debugger to connections from `localhost`).

int **network/debug/remote_port** 🔗

The port to listen to when starting the remote debugger. Godot will try to use port numbers above the configured number if the configured number is already taken by another application.

String **network/http_proxy/host** 🔗

The host to use to contact the HTTP and HTTPS proxy in the editor (for the asset library and export template downloads). See also network/http_proxy/port.

Note: Godot currently doesn't automatically use system proxy settings, so you have to enter them manually here if needed.

int **network/http_proxy/port** 🔗

The port number to use to contact the HTTP and HTTPS proxy in the editor (for the asset library and export template downloads). See also network/http_proxy/host.

Note: Godot currently doesn't automatically use system proxy settings, so you have to enter them manually here if needed.

String **network/tls/editor_tls_certificates** 🔗

The TLS certificate bundle to use for HTTP requests made within the editor (e.g. from the AssetLib tab). If left empty, the included Mozilla certificate bundle [https://github.com/godotengine/godot/blob/master/thirdparty/certs/ca-certificates.crt] will be used.

String **project_manager/default_renderer** 🔗

The renderer type that will be checked off by default when creating a new project. Accepted strings are "forward_plus", "mobile" or "gl_compatibility".

---

int **project_manager/directory_naming_convention** 🔗

Directory naming convention for the project manager. Options are "No convention" (project name is directory name), "kebab-case" (default), "snake_case", "camelCase", "PascalCase", or "Title Case".

---

int **project_manager/sorting_order** 🔗

The sorting order to use in the project manager. When changing the sorting order in the project manager, this setting is set permanently in the editor settings.

---

bool **run/auto_save/save_before_running** 🔗

If `true`, saves all scenes and scripts automatically before running the project. Setting this to `false` prevents the editor from saving if there are no changes which can speed up the project startup slightly, but it makes it possible to run a project that has unsaved changes. (Unsaved changes will not be visible in the running project.)

---

int **run/bottom_panel/action_on_play** 🔗

The action to execute on the bottom panel when running the project.

---

int **run/bottom_panel/action_on_stop** 🔗

The action to execute on the bottom panel when stopping the project.

---

bool **run/output/always_clear_output_on_play** 🔗

If `true`, the editor will clear the Output panel when running the project.

---

int **run/output/font_size** 🔗

The size of the font in the **Output** panel at the bottom of the editor. This setting does not impact the font size of the script editor (see interface/editor/code_font_size).

---

int **run/output/max_lines** 🔗

Maximum number of lines to show at any one time in the Output panel.

---

bool **run/platforms/linuxbsd/prefer_wayland** 🔗

If `true`, on Linux/BSD, the editor will check for Wayland first instead of X11 (if available).

---

int **run/window_placement/android_window** 🔗

The Android window to display the project on when starting the project from the editor.

Note: Only available in the Android editor.

---

int **run/window_placement/rect** 🔗

The window mode to use to display the project when starting the project from the editor.

Vector2 **run/window_placement/rect_custom_position** 🔗

The custom position to use when starting the project from the editor (in pixels from the top-left corner). Only effective if run/window_placement/rect is set to **Custom Position**.

---

int **run/window_placement/screen** 🔗

The monitor to display the project on when starting the project from the editor.

---

bool **text_editor/appearance/caret/caret_blink** 🔗

If `true`, makes the caret blink according to text_editor/appearance/caret/caret_blink_interval. Disabling this setting can improve battery life on laptops if you spend long amounts of time in the script editor, since it will reduce the frequency at which the editor needs to be redrawn.

---

float **text_editor/appearance/caret/caret_blink_interval** 🔗

The interval at which the caret will blink (in seconds). See also text_editor/appearance/caret/caret_blink.

---

bool **text_editor/appearance/caret/highlight_all_occurrences** 🔗

If `true`, highlights all occurrences of the currently selected text in the script editor. See also text_editor/theme/highlighting/word_highlighted_color.

---

bool **text_editor/appearance/caret/highlight_current_line** 🔗

If `true`, colors the background of the line the caret is currently on with text_editor/theme/highlighting/current_line_color.

---

int **text_editor/appearance/caret/type** 🔗

The shape of the caret to use in the script editor. **Line** displays a vertical line to the left of the current character, whereas **Block** displays an outline over the current character.

---

int **text_editor/appearance/guidelines/line_length_guideline_hard_column** 🔗

The column at which to display a subtle line as a line length guideline for scripts. This should generally be greater than text_editor/appearance/guidelines/line_length_guideline_soft_column.

---

int **text_editor/appearance/guidelines/line_length_guideline_soft_column**

The column at which to display a *very* subtle line as a line length guideline for scripts. This should generally be lower than text_editor/appearance/guidelines/line_length_guideline_hard_column.

bool **text_editor/appearance/guidelines/show_line_length_guidelines**

If `true`, displays line length guidelines to help you keep line lengths in check. See also text_editor/appearance/guidelines/line_length_guideline_soft_column and text_editor/appearance/guidelines/line_length_guideline_hard_column.

bool **text_editor/appearance/gutters/highlight_type_safe_lines**

If `true`, highlights type-safe lines by displaying their line number color with text_editor/theme/highlighting/safe_line_number_color instead of text_editor/theme/highlighting/line_number_color. Type-safe lines are lines of code where the type of all variables is known at compile-time. These type-safe lines may run faster thanks to typed instructions.

bool **text_editor/appearance/gutters/line_numbers_zero_padded**

If `true`, displays line numbers with zero padding (e.g. `007` instead of `7`).

bool **text_editor/appearance/gutters/show_info_gutter**

If `true`, displays a gutter at the left containing icons for methods with signal connections and for overridden methods.

bool **text_editor/appearance/gutters/show_line_numbers**

If `true`, displays line numbers in a gutter at the left.

int **text_editor/appearance/lines/autowrap_mode**

If text_editor/appearance/lines/word_wrap is set to `1`, sets text wrapping mode. To see how each mode behaves, see AutowrapMode.

bool **text_editor/appearance/lines/code_folding**

If `true`, displays the folding arrows next to indented code sections and allows code folding. If `false`, hides the folding arrows next to indented code sections and disallows code folding.

int **text_editor/appearance/lines/word_wrap**

If `true`, wraps long lines over multiple lines to avoid horizontal scrolling. This is a display-only feature; it does not actually insert line breaks in your scripts.

int **text_editor/appearance/minimap/minimap_width**

The width of the minimap in the script editor (in pixels).

bool **text_editor/appearance/minimap/show_minimap**

If `true`, draws an overview of the script near the scroll bar. The minimap can be left-clicked to scroll directly to a location in an "absolute" manner.

bool **text_editor/appearance/whitespace/draw_spaces**

If `true`, draws space characters as centered points.

bool **text_editor/appearance/whitespace/draw_tabs**

If `true`, draws tab characters as chevrons.

int **text_editor/appearance/whitespace/line_spacing**

The space to add between lines (in pixels). Greater line spacing can help improve readability at the cost of displaying fewer lines on screen.

bool **text_editor/behavior/files/auto_reload_scripts_on_external_change**

If `true`, automatically reloads scripts in the editor when they have been modified and saved by external editors.

int **text_editor/behavior/files/autosave_interval_secs**

If set to a value greater than `0`, automatically saves the current script following the specified interval (in seconds). This can be used to prevent data loss if the editor crashes.

bool **text_editor/behavior/files/convert_indent_on_save**

If `true`, converts indentation to match the script editor's indentation settings when saving a script. See also text_editor/behavior/indent/type.

bool **text_editor/behavior/files/restore_scripts_on_load**

If `true`, reopens scripts that were opened in the last session when the editor is reopened on a given project.

bool **text_editor/behavior/files/trim_final_newlines_on_save**

If `true`, trims all empty newlines after the final newline when saving a script. Final newlines refer to the empty newlines found at the end of files. Since these serve no practical purpose, they can and should be removed to make version control diffs less noisy.

bool **text_editor/behavior/files/trim_trailing_whitespace_on_save**

If `true`, trims trailing whitespace when saving a script. Trailing whitespace refers to tab and space characters placed at the end of lines. Since these serve no practical purpose, they can and should be removed to make version control diffs less noisy.

bool **text_editor/behavior/indent/auto_indent**

If `true`, automatically indents code when pressing the `Enter` key based on blocks above the new line.

bool **text_editor/behavior/indent/indent_wrapped_lines**

If `true`, all wrapped lines are indented to the same amount as the unwrapped line.

int **text_editor/behavior/indent/size**

When using tab indentation, determines the length of each tab. When using space indentation, determines how many spaces are inserted when pressing `Tab` and when automatic indentation is performed.

int **text_editor/behavior/indent/type**

The indentation style to use (tabs or spaces).

Note: The GDScript style guide recommends using tabs for indentation. It is advised to change this setting only if you need to work on a project that currently uses spaces for indentation.

String **text_editor/behavior/navigation/custom_word_separators**

The characters to consider as word delimiters if text_editor/behavior/navigation/use_custom_word_separators is `true`. This is in addition to default characters if text_editor/behavior/navigation/use_default_word_separators is `true`. The characters should be defined without separation, for example `_♥=`.

bool **text_editor/behavior/navigation/drag_and_drop_selection**

If `true`, allows drag-and-dropping text in the script editor to move text. Disable this if you find yourself accidentally drag-and-dropping text in the script editor.

bool **text_editor/behavior/navigation/move_caret_on_right_click**

If `true`, the caret will be moved when right-clicking somewhere in the script editor (like when left-clicking or middle-clicking). If `false`, the caret will only be moved when left-clicking or middle-clicking somewhere.

bool **text_editor/behavior/navigation/open_script_when_connecting_signal_to_existing_method**

If `true`, opens the script editor when connecting a signal to an existing script method from the Node dock.

bool **text_editor/behavior/navigation/scroll_past_end_of_file**

If `true`, allows scrolling past the end of the file.

bool **text_editor/behavior/navigation/smooth_scrolling**

If `true`, enables a smooth scrolling animation when using the mouse wheel to scroll. See text_editor/behavior/navigation/v_scroll_speed for the speed of this animation.

Note: text_editor/behavior/navigation/smooth_scrolling currently behaves poorly in projects where ProjectSettings.physics/common/physics_ticks_per_second has been increased significantly from its default value (`60`). In this case, it is recommended to disable this setting.

bool **text_editor/behavior/navigation/stay_in_script_editor_on_node_selected**

If `true`, prevents automatically switching between the Script and 2D/3D screens when selecting a node in the Scene tree dock.

bool **text_editor/behavior/navigation/use_custom_word_separators**

If `true`, uses the characters in text_editor/behavior/navigation/custom_word_separators as word separators for word navigation and operations. This is in addition to the default characters if text_editor/behavior/navigation/use_default_word_separators is also enabled. Word navigation and operations include double-clicking on a word or holding `Ctrl` (`Cmd` on macOS) while pressing `left`, `right`, `backspace`, or `delete`.

bool **text_editor/behavior/navigation/use_default_word_separators**

If `true`, uses the characters in `` `!"#$%&'()*+,-./:;<=>?@[\]^`{|}~ ``, the Unicode General Punctuation table, and the Unicode CJK Punctuation table as word separators for word navigation and operations. If `false`, a subset of these characters are used and does not include the characters `<>$~^=+|`. This is in addition to custom characters if text_editor/behavior/navigation/use_custom_word_separators is also enabled. These characters are used to determine where a word stops. Word navigation and operations include double-clicking on a word or holding `Ctrl` (`Cmd` on macOS) while pressing `left`, `right`, `backspace`, or `delete`.

int **text_editor/behavior/navigation/v_scroll_speed**

The speed of scrolling in lines per second when text_editor/behavior/navigation/smooth_scrolling is `true`. Higher values make the script scroll by faster when using the mouse wheel.

Note: You can hold down `Alt` while using the mouse wheel to temporarily scroll 5 times faster.

---

bool **text_editor/completion/add_node_path_literals** 🔗

If `true`, uses NodePath instead of String when appropriate for code autocompletion or for drag and dropping object properties into the script editor.

---

bool **text_editor/completion/add_string_name_literals** 🔗

If `true`, uses StringName instead of String when appropriate for code autocompletion.

---

bool **text_editor/completion/add_type_hints** 🔗

If `true`, adds GDScript static typing hints such as `-> void` and `: int` when using code autocompletion or when creating onready variables by drag and dropping nodes into the script editor while pressing the `Ctrl` key. If `true`, newly created scripts will also automatically have type hints added to their method parameters and return types.

---

bool **text_editor/completion/auto_brace_complete** 🔗

If `true`, automatically completes braces when making use of code completion.

---

float **text_editor/completion/code_complete_delay** 🔗

The delay in seconds after which autocompletion suggestions should be displayed when the user stops typing.

---

bool **text_editor/completion/code_complete_enabled** 🔗

If `true`, code completion will be triggered automatically after text_editor/completion/code_complete_delay. Even if `false`, code completion can be triggered manually with the `ui_text_completion_query` action (by default `Ctrl + Space` or `Cmd + Space` on macOS).

---

bool **text_editor/completion/colorize_suggestions** 🔗

If `true` enables the coloring for some items in the autocompletion suggestions, like vector components.

---

bool **text_editor/completion/complete_file_paths** 🔗

If `true`, provides autocompletion suggestions for file paths in methods such as `load()` and `preload()`.

---

float **text_editor/completion/idle_parse_delay** 🔗

The delay in seconds after which the script editor should check for errors when the user stops typing.

---

bool **text_editor/completion/put_callhint_tooltip_below_current_line** 🔗

If `true`, the code completion tooltip will appear below the current line unless there is no space on screen below the current line. If `false`, the code completion tooltip will appear above the current line.

---

bool **text_editor/completion/use_single_quotes** 🔗

If `true`, performs string autocompletion with single quotes. If `false`, performs string autocompletion with double quotes (which matches the GDScript style guide).

---

int **text_editor/help/class_reference_examples** 🔗

Controls which multi-line code blocks should be displayed in the editor help. This setting does not affect single-line code literals in the editor help.

---

int **text_editor/help/help_font_size** 🔗

The font size to use for the editor help (built-in class reference).

---

int **text_editor/help/help_source_font_size** 🔗

The font size to use for code samples in the editor help (built-in class reference).

---

int **text_editor/help/help_title_font_size** 🔗

The font size to use for headings in the editor help (built-in class reference).

---

bool **text_editor/help/show_help_index** 🔗

If `true`, displays a table of contents at the left of the editor help (at the location where the members overview would appear when editing a script).

---

bool **text_editor/script_list/show_members_overview** 🔗

If `true`, displays an overview of the current script's member variables and functions at the left of the script editor. See also text_editor/script_list/sort_members_outline_alphabetically.

---

bool **text_editor/script_list/sort_members_outline_alphabetically** 🔗

If `true`, sorts the members outline (located at the left of the script editor) using alphabetical order. If `false`, sorts the members outline depending on the order in which members are found in the script.

Note: Only effective if text_editor/script_list/show_members_overview is `true`.

String **text_editor/theme/color_theme** 🔗

The syntax theme to use in the script editor.

You can save your own syntax theme from your current settings by using **File > Theme > Save As...** at the top of the script editor. The syntax theme will then be available locally in the list of color themes.

You can find additional syntax themes to install in the godot-syntax-themes [https://github.com/godotengine/godot-syntax-themes] repository.

Color **text_editor/theme/highlighting/background_color**

The script editor's background color. If set to a translucent color, the editor theme's base color will be visible behind.

Color **text_editor/theme/highlighting/base_type_color**

The script editor's base type color (used for types like Vector2, Vector3, Color, ...).

Color **text_editor/theme/highlighting/bookmark_color**

The script editor's bookmark icon color (displayed in the gutter).

Color **text_editor/theme/highlighting/brace_mismatch_color**

The script editor's brace mismatch color. Used when the caret is currently on a mismatched brace, parenthesis or bracket character.

Color **text_editor/theme/highlighting/breakpoint_color**

The script editor's breakpoint icon color (displayed in the gutter).

Color **text_editor/theme/highlighting/caret_background_color**

The script editor's caret background color.

Note: This setting has no effect as it's currently unused.

Color **text_editor/theme/highlighting/caret_color**

The script editor's caret color.

Color **text_editor/theme/highlighting/code_folding_color**

The script editor's color for the code folding icon (displayed in the gutter).

Color **text_editor/theme/highlighting/comment_color**

The script editor's comment color.

Note: In GDScript, unlike Python, multiline strings are not considered to be comments, and will use the string highlighting color instead.

Color **text_editor/theme/highlighting/completion_background_color** 🔗

The script editor's autocompletion box background color.

Color **text_editor/theme/highlighting/completion_existing_color** 🔗

The script editor's autocompletion box background color to highlight existing characters in the completion results. This should be a translucent color so that text_editor/theme/highlighting/completion_selected_color can be seen behind.

Color **text_editor/theme/highlighting/completion_font_color** 🔗

The script editor's autocompletion box text color.

Color **text_editor/theme/highlighting/completion_scroll_color** 🔗

The script editor's autocompletion box scroll bar color.

Color **text_editor/theme/highlighting/completion_scroll_hovered_color** 🔗

The script editor's autocompletion box scroll bar color when hovered or pressed with the mouse.

Color **text_editor/theme/highlighting/completion_selected_color** 🔗

The script editor's autocompletion box background color for the currently selected line.

Color **text_editor/theme/highlighting/control_flow_keyword_color** 🔗

The script editor's control flow keyword color (used for keywords like `if`, `for`, `return`, ...).

Color **text_editor/theme/highlighting/current_line_color** 🔗

The script editor's background color for the line the caret is currently on. This should be set to a translucent color so that it can display on top of other line color modifiers such as text_editor/theme/highlighting/mark_color.

Color **text_editor/theme/highlighting/doc_comment_color** 🔗

The script editor's documentation comment color. In GDScript, this is used for comments starting with `##`. In C#, this is used for comments starting with `///` or `/**`.

Color **text_editor/theme/highlighting/engine_type_color** 🔗

The script editor's engine type color (Vector2, Vector3, Color, ...).

Color **text_editor/theme/highlighting/executing_line_color**

The script editor's color for the debugger's executing line icon (displayed in the gutter).

Color **text_editor/theme/highlighting/folded_code_region_color**

The script editor's background line highlighting color for folded code region.

Color **text_editor/theme/highlighting/function_color**

The script editor's function call color.

Note: When using the GDScript syntax highlighter, this is replaced by the function definition color configured in the syntax theme for function definitions (e.g. `func _ready():`).

Color **text_editor/theme/highlighting/keyword_color**

The script editor's non-control flow keyword color (used for keywords like `var`, `func`, `extends`, ...).

Color **text_editor/theme/highlighting/line_length_guideline_color**

The script editor's color for the line length guideline. The "hard" line length guideline will be drawn with this color, whereas the "soft" line length guideline will be drawn with half of its opacity.

Color **text_editor/theme/highlighting/line_number_color**

The script editor's color for line numbers. See also text_editor/theme/highlighting/safe_line_number_color.

Color **text_editor/theme/highlighting/mark_color**

The script editor's background color for lines with errors. This should be set to a translucent color so that it can display on top of other line color modifiers such as text_editor/theme/highlighting/current_line_color.

Color **text_editor/theme/highlighting/member_variable_color**

The script editor's color for member variables on objects (e.g. `self.some_property`).

Note: This color is not used for local variable declaration and access.

Color **text_editor/theme/highlighting/number_color**

The script editor's color for numbers (integer and floating-point).

Color **text_editor/theme/highlighting/safe_line_number_color**

The script editor's color for type-safe line numbers. See also text_editor/theme/highlighting/line_number_color.

Note: Only displayed if text_editor/appearance/gutters/highlight_type_safe_lines is `true`.

Color **text_editor/theme/highlighting/search_result_border_color**

The script editor's color for the border of search results. This border helps bring further attention to the search result. Set this color's opacity to 0 to disable the border.

Color **text_editor/theme/highlighting/search_result_color**

The script editor's background color for search results.

Color **text_editor/theme/highlighting/selection_color**

The script editor's background color for the currently selected text.

Color **text_editor/theme/highlighting/string_color**

The script editor's color for strings (single-line and multi-line).

Color **text_editor/theme/highlighting/symbol_color**

The script editor's color for operators (`( ) [ ] { } + - * /`, ...).

Color **text_editor/theme/highlighting/text_color**

The script editor's color for text not highlighted by any syntax highlighting rule.

Color **text_editor/theme/highlighting/text_selected_color**

The script editor's background color for text. This should be set to a translucent color so that it can display on top of other line color modifiers such as text_editor/theme/highlighting/current_line_color.

Color **text_editor/theme/highlighting/user_type_color**

The script editor's color for user-defined types (using `class_name`).

Color **text_editor/theme/highlighting/word_highlighted_color**

The script editor's color for words highlighted by selecting them. Only visible if text_editor/appearance/caret/highlight_all_occurrences is `true`.

Method Descriptions

void **add_property_info**(info: Dictionary)

Adds a custom property info to a property. The dictionary must contain:

- `name`: String (the name of the property)
- `type`: int (see Variant.Type)
- optionally `hint`: int (see PropertyHint) and `hint_string`: String

Example:

GDScript

```
var settings = EditorInterface.get_editor_settings()
settings.set("category/property_name", 0)

var property_info = {
    "name": "category/property_name",
    "type": TYPE_INT,
    "hint": PROPERTY_HINT_ENUM,
    "hint_string": "one,two,three"
}

settings.add_property_info(property_info)
```

C#

```
var settings = GetEditorInterface().GetEditorSettings();
settings.Set("category/property_name", 0);

var propertyInfo = new Godot.Collections.Dictionary
{
    {"name", "category/propertyName"},
    {"type", Variant.Type.Int},
    {"hint", PropertyHint.Enum},
    {"hint_string", "one,two,three"}
};

settings.AddPropertyInfo(propertyInfo);
```

---

bool **check_changed_settings_in_group**(setting_prefix: String) const

Checks if any settings with the prefix `setting_prefix` exist in the set of changed settings. See also get_changed_settings.

---

void **erase**(property: String)

Erases the setting whose name is specified by `property`.

PackedStringArray **get_changed_settings**() const

Gets an array of the settings which have been changed since the last save. Note that internally `changed_settings` is cleared after a successful save, so generally the most appropriate place to use this method is when processing NOTIFICATION_EDITOR_SETTINGS_CHANGED.

PackedStringArray **get_favorites**() const

Returns the list of favorite files and directories for this project.

Variant **get_project_metadata**(section: String, key: String, default: Variant = null) const

Returns project-specific metadata for the `section` and `key` specified. If the metadata doesn't exist, `default` will be returned instead. See also set_project_metadata.

PackedStringArray **get_recent_dirs**() const

Returns the list of recently visited folders in the file dialog for this project.

Variant **get_setting**(name: String) const

Returns the value of the setting specified by `name`. This is equivalent to using Object.get on the EditorSettings instance.

---

bool **has_setting**(name: String) const

Returns `true` if the setting specified by `name` exists, `false` otherwise.

---

void **mark_setting_changed**(setting: String)

Marks the passed editor setting as being changed, see get_changed_settings. Only settings which exist (see has_setting) will be accepted.

---

void **set_builtin_action_override**(name: String, actions_list: Array[InputEvent])

Overrides the built-in editor action `name` with the input actions defined in `actions_list`.

---

void **set_favorites**(dirs: PackedStringArray)

Sets the list of favorite files and directories for this project.

---

void **set_initial_value**(name: StringName, value: Variant, update_current: bool)

Sets the initial value of the setting specified by `name` to `value`. This is used to provide a value for the Revert button in the Editor Settings. If `update_current` is true, the current value of the setting will be set to `value` as well.

---

void **set_project_metadata**(section: String, key: String, data: Variant)

Sets project-specific metadata with the `section`, `key` and `data` specified. This metadata is stored outside the project folder and therefore won't be checked into version control. See also get_project_metadata.

---

void **set_recent_dirs**(dirs: PackedStringArray) 🔗

Sets the list of recently visited folders in the file dialog for this project.

---

void **set_setting**(name: String, value: Variant) 🔗

Sets the `value` of the setting specified by `name`. This is equivalent to using Object.set on the EditorSettings instance.

EditorSyntaxHighlighter

Inherits: SyntaxHighlighter **<** Resource **<** RefCounted **<** Object

Base class for SyntaxHighlighter used by the ScriptEditor.

Description

Base class that all SyntaxHighlighters used by the ScriptEditor extend from.

Add a syntax highlighter to an individual script by calling ScriptEditorBase.add_syntax_highlighter. To apply to all scripts on open, call ScriptEditor.register_syntax_highlighter.

Methods

String	_get_name() virtual const
PackedStringArray	_get_supported_languages() virtual const

Method Descriptions

String **_get_name**() virtual const 🔗

Virtual method which can be overridden to return the syntax highlighter name.

PackedStringArray **_get_supported_languages**() virtual const 🔗

Virtual method which can be overridden to return the supported language names.

Environment

Inherits: Resource **<** RefCounted **<** Object

Resource for environment nodes (like WorldEnvironment) that define multiple rendering options.

Description

Resource for environment nodes (like WorldEnvironment) that define multiple environment operations (such as background Sky or Color, ambient light, fog, depth-of-field...). These parameters affect the final render of the scene. The order of these operations is:

- Depth of Field Blur
- Glow
- Tonemap (Auto Exposure)
- Adjustments

Tutorials

- Environment and post-processing
- High dynamic range lighting
- 3D Material Testers Demo [https://godotengine.org/asset-library/asset/2742]
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

float	adjustment_brightness	`1.0`
Texture	adjustment_color_correction	
float	adjustment_contrast	`1.0`

bool	adjustment_enabled	`false`
float	adjustment_saturation	`1.0`
Color	ambient_light_color	`Color(0, 0, 0, 1)`
float	ambient_light_energy	`1.0`
float	ambient_light_sky_contribution	`1.0`
AmbientSource	ambient_light_source	`0`
int	background_camera_feed_id	`1`
int	background_canvas_max_layer	`0`
Color	background_color	`Color(0, 0, 0, 1)`
float	background_energy_multiplier	`1.0`
float	background_intensity	`30000.0`
BGMode	background_mode	`0`
float	fog_aerial_perspective	`0.0`
float	fog_density	`0.01`

float	fog_depth_begin	`10.0`
float	fog_depth_curve	`1.0`
float	fog_depth_end	`100.0`
bool	fog_enabled	`false`
float	fog_height	`0.0`
float	fog_height_density	`0.0`
Color	fog_light_color	`Color(0.518, 0.553, 0.608, 1)`
float	fog_light_energy	`1.0`
FogMode	fog_mode	`0`
float	fog_sky_affect	`1.0`
float	fog_sun_scatter	`0.0`
GlowBlendMode	glow_blend_mode	`2`
float	glow_bloom	`0.0`
bool	glow_enabled	`false`

float	glow_hdr_luminance_cap	`12.0`
float	glow_hdr_scale	`2.0`
float	glow_hdr_threshold	`1.0`
float	glow_intensity	`0.8`
float	glow_levels/1	`0.0`
float	glow_levels/2	`0.0`
float	glow_levels/3	`1.0`
float	glow_levels/4	`0.0`
float	glow_levels/5	`1.0`
float	glow_levels/6	`0.0`
float	glow_levels/7	`0.0`
Texture	glow_map	
float	glow_map_strength	`0.8`
float	glow_mix	`0.05`

bool	glow_normalized	`false`
float	glow_strength	`1.0`
ReflectionSource	reflected_light_source	`0`
float	sdfgi_bounce_feedback	`0.5`
float	sdfgi_cascade0_distance	`12.8`
int	sdfgi_cascades	`4`
bool	sdfgi_enabled	`false`
float	sdfgi_energy	`1.0`
float	sdfgi_max_distance	`204.8`
float	sdfgi_min_cell_size	`0.2`
float	sdfgi_normal_bias	`1.1`
float	sdfgi_probe_bias	`1.1`
bool	sdfgi_read_sky_light	`true`
bool	sdfgi_use_occlusion	`false`
SDFGIYScale	sdfgi_y_scale	`1`

Sky	sky	
float	sky_custom_fov	`0.0`
Vector3	sky_rotation	`Vector3(0, 0, 0)`
float	ssao_ao_channel_affect	`0.0`
float	ssao_detail	`0.5`
bool	ssao_enabled	`false`
float	ssao_horizon	`0.06`
float	ssao_intensity	`2.0`
float	ssao_light_affect	`0.0`
float	ssao_power	`1.5`
float	ssao_radius	`1.0`
float	ssao_sharpness	`0.98`
bool	ssil_enabled	`false`
float	ssil_intensity	`1.0`

float	ssil_normal_rejection	`1.0`
float	ssil_radius	`5.0`
float	ssil_sharpness	`0.98`
float	ssr_depth_tolerance	`0.2`
bool	ssr_enabled	`false`
float	ssr_fade_in	`0.15`
float	ssr_fade_out	`2.0`
int	ssr_max_steps	`64`
float	tonemap_exposure	`1.0`
ToneMapper	tonemap_mode	`0`
float	tonemap_white	`1.0`
Color	volumetric_fog_albedo	`Color(1, 1, 1, 1)`
float	volumetric_fog_ambient_inject	`0.0`
float	volumetric_fog_anisotropy	`0.2`

float	volumetric_fog_density	`0.05`
float	volumetric_fog_detail_spread	`2.0`
Color	volumetric_fog_emission	`Color(0, 0, 0, 1)`
float	volumetric_fog_emission_energy	`1.0`
bool	volumetric_fog_enabled	`false`
float	volumetric_fog_gi_inject	`1.0`
float	volumetric_fog_length	`64.0`
float	volumetric_fog_sky_affect	`1.0`
float	volumetric_fog_temporal_reprojection_amount	`0.9`
bool	volumetric_fog_temporal_reprojection_enabled	`true`

Methods

float	get_glow_level(idx: int) const
void	set_glow_level(idx: int, intensity: float)

Enumerations

enum **BGMode**: 🔗

BGMode **BG_CLEAR_COLOR** = `0`

Clears the background using the clear color defined in ProjectSettings.rendering/environment/defaults/default_clear_color.

BGMode **BG_COLOR** = `1`

Clears the background using a custom clear color.

BGMode **BG_SKY** = `2`

Displays a user-defined sky in the background.

BGMode **BG_CANVAS** = `3`

Displays a CanvasLayer in the background.

BGMode **BG_KEEP** = `4`

Keeps on screen every pixel drawn in the background. This is the fastest background mode, but it can only be safely used in fully-interior scenes (no visible sky or sky reflections). If enabled in a scene where the background is visible, "ghost trail" artifacts will be visible when moving the camera.

BGMode **BG_CAMERA_FEED** = `5`

Displays a camera feed in the background.

BGMode **BG_MAX** = `6`

Represents the size of the BGMode enum.

---

enum **AmbientSource**: 🔗

AmbientSource **AMBIENT_SOURCE_BG** = `0`

Gather ambient light from whichever source is specified as the background.

AmbientSource **AMBIENT_SOURCE_DISABLED** = `1`

Disable ambient light. This provides a slight performance boost over AMBIENT_SOURCE_SKY.

AmbientSource **AMBIENT_SOURCE_COLOR** = `2`

Specify a specific Color for ambient light. This provides a slight performance boost over AMBIENT_SOURCE_SKY.

AmbientSource **AMBIENT_SOURCE_SKY** = `3`

Gather ambient light from the Sky regardless of what the background is.

---

enum **ReflectionSource**: 🔗

ReflectionSource **REFLECTION_SOURCE_BG** = `0`

Use the background for reflections.

ReflectionSource **REFLECTION_SOURCE_DISABLED** = `1`

Disable reflections. This provides a slight performance boost over other options.

ReflectionSource **REFLECTION_SOURCE_SKY** = `2`

Use the Sky for reflections regardless of what the background is.

---

enum **ToneMapper**: 🔗

ToneMapper **TONE_MAPPER_LINEAR** = `0`

Linear tonemapper operator. Reads the linear data and passes it on unmodified. This can cause bright lighting to look blown out, with noticeable clipping in the output colors.

ToneMapper **TONE_MAPPER_REINHARDT** = `1`

Reinhardt tonemapper operator. Performs a variation on rendered pixels' colors by this formula: `color = color / (1 + color)`. This avoids clipping bright highlights, but the resulting image can look a bit dull.

ToneMapper **TONE_MAPPER_FILMIC** = `2`

Filmic tonemapper operator. This avoids clipping bright highlights, with a resulting image that usually looks more vivid than TONE_MAPPER_REINHARDT.

ToneMapper **TONE_MAPPER_ACES** = `3`

Use the Academy Color Encoding System tonemapper. ACES is slightly more expensive than other options, but it handles bright lighting in a more realistic fashion by desaturating it as it becomes brighter. ACES typically has a more contrasted output compared to TONE_MAPPER_REINHARDT and TONE_MAPPER_FILMIC.

Note: This tonemapping operator is called "ACES Fitted" in Godot 3.x.

enum **GlowBlendMode**: 🔗

GlowBlendMode **GLOW_BLEND_MODE_ADDITIVE** = `0`

Additive glow blending mode. Mostly used for particles, glows (bloom), lens flare, bright sources.

GlowBlendMode **GLOW_BLEND_MODE_SCREEN** = `1`

Screen glow blending mode. Increases brightness, used frequently with bloom.

GlowBlendMode **GLOW_BLEND_MODE_SOFTLIGHT** = `2`

Soft light glow blending mode. Modifies contrast, exposes shadows and highlights (vivid bloom).

GlowBlendMode **GLOW_BLEND_MODE_REPLACE** = `3`

Replace glow blending mode. Replaces all pixels' color by the glow value. This can be used to simulate a full-screen blur effect by tweaking the glow parameters to match the original image's brightness.

GlowBlendMode **GLOW_BLEND_MODE_MIX** = `4`

Mixes the glow with the underlying color to avoid increasing brightness as much while still maintaining a glow effect.

enum **FogMode**: 🔗

FogMode **FOG_MODE_EXPONENTIAL** = `0`

Use a physically-based fog model defined primarily by fog density.

FogMode **FOG_MODE_DEPTH** = `1`

Use a simple fog model defined by start and end positions and a custom curve. While not physically accurate, this model can be useful when you need more artistic control.

---

enum **SDFGIYScale**: 🔗

SDFGIYScale **SDFGI_Y_SCALE_50_PERCENT** = `0`

Use 50% scale for SDFGI on the Y (vertical) axis. SDFGI cells will be twice as short as they are wide. This allows providing increased GI detail and reduced light leaking with thin floors and ceilings. This is usually the best choice for scenes that don't feature much verticality.

SDFGIYScale **SDFGI_Y_SCALE_75_PERCENT** = `1`

Use 75% scale for SDFGI on the Y (vertical) axis. This is a balance between the 50% and 100% SDFGI Y scales.

SDFGIYScale **SDFGI_Y_SCALE_100_PERCENT** = `2`

Use 100% scale for SDFGI on the Y (vertical) axis. SDFGI cells will be as tall as they are wide. This is usually the best choice for highly vertical scenes. The downside is that light leaking may become more noticeable with thin floors and ceilings.

Property Descriptions

float **adjustment_brightness** = `1.0` 🔗

- void **set_adjustment_brightness**(value: float)
- float **get_adjustment_brightness**()

The global brightness value of the rendered scene. Effective only if adjustment_enabled is `true`.

Texture **adjustment_color_correction**

- void **set_adjustment_color_correction**(value: Texture)
- Texture **get_adjustment_color_correction**()

The Texture2D or Texture3D lookup table (LUT) to use for the built-in post-process color grading. Can use a GradientTexture1D for a 1-dimensional LUT, or a Texture3D for a more complex LUT. Effective only if adjustment_enabled is `true`.

---

float **adjustment_contrast** = `1.0`

- void **set_adjustment_contrast**(value: float)
- float **get_adjustment_contrast**()

The global contrast value of the rendered scene (default value is 1). Effective only if adjustment_enabled is `true`.

---

bool **adjustment_enabled** = `false`

- void **set_adjustment_enabled**(value: bool)
- bool **is_adjustment_enabled**()

If `true`, enables the `adjustment_*` properties provided by this resource. If `false`, modifications to the `adjustment_*` properties will have no effect on the rendered scene.

---

float **adjustment_saturation** = `1.0`

- void **set_adjustment_saturation**(value: float)
- float **get_adjustment_saturation**()

The global color saturation value of the rendered scene (default value is 1). Effective only if adjustment_enabled is `true`.

Color **ambient_light_color** = `Color(0, 0, 0, 1)`

- void **set_ambient_light_color**(value: Color)
- Color **get_ambient_light_color**()

The ambient light's Color. Only effective if ambient_light_sky_contribution is lower than `1.0` (exclusive).

---

float **ambient_light_energy** = `1.0`

- void **set_ambient_light_energy**(value: float)
- float **get_ambient_light_energy**()

The ambient light's energy. The higher the value, the stronger the light. Only effective if ambient_light_sky_contribution is lower than `1.0` (exclusive).

---

float **ambient_light_sky_contribution** = `1.0`

- void **set_ambient_light_sky_contribution**(value: float)
- float **get_ambient_light_sky_contribution**()

Defines the amount of light that the sky brings on the scene. A value of `0.0` means that the sky's light emission has no effect on the scene illumination, thus all ambient illumination is provided by the ambient light. On the contrary, a value of `1.0` means that *all* the light that affects the scene is provided by the sky, thus the ambient light parameter has no effect on the scene.

Note: ambient_light_sky_contribution is internally clamped between `0.0` and `1.0` (inclusive).

AmbientSource **ambient_light_source** = `0`

- void **set_ambient_source**(value: AmbientSource)
- AmbientSource **get_ambient_source**()

The ambient light source to use for rendering materials and global illumination.

---

int **background_camera_feed_id** = `1`

- void **set_camera_feed_id**(value: int)
- int **get_camera_feed_id**()

The ID of the camera feed to show in the background.

---

int **background_canvas_max_layer** = `0`

- void **set_canvas_max_layer**(value: int)
- int **get_canvas_max_layer**()

The maximum layer ID to display. Only effective when using the BG_CANVAS background mode.

Color **background_color** = `Color(0, 0, 0, 1)`

- void **set_bg_color**(value: Color)
- Color **get_bg_color**()

The Color displayed for clear areas of the scene. Only effective when using the BG_COLOR background mode.

---

float **background_energy_multiplier** = `1.0`

- void **set_bg_energy_multiplier**(value: float)
- float **get_bg_energy_multiplier**()

Multiplier for background energy. Increase to make background brighter, decrease to make background dimmer.

---

float **background_intensity** = `30000.0`

- void **set_bg_intensity**(value: float)
- float **get_bg_intensity**()

Luminance of background measured in nits (candela per square meter). Only used when ProjectSettings.rendering/lights_and_shadows/use_physical_light_units is enabled. The default value is roughly equivalent to the sky at midday.

BGMode **background_mode** = `0`

- void **set_background**(value: BGMode)
- BGMode **get_background**()

The background mode. See BGMode for possible values.

---

float **fog_aerial_perspective** = `0.0`

- void **set_fog_aerial_perspective**(value: float)
- float **get_fog_aerial_perspective**()

If set above `0.0` (exclusive), blends between the fog's color and the color of the background Sky. This has a small performance cost when set above `0.0`. Must have background_mode set to BG_SKY.

This is useful to simulate aerial perspective [https://en.wikipedia.org/wiki/Aerial_perspective] in large scenes with low density fog. However, it is not very useful for high-density fog, as the sky will shine through. When set to `1.0`, the fog color comes completely from the Sky. If set to `0.0`, aerial perspective is disabled.

---

float **fog_density** = `0.01`

- void **set_fog_density**(value: float)
- float **get_fog_density**()

The fog density to be used. This is demonstrated in different ways depending on the fog_mode mode chosen:

Exponential Fog Mode: Higher values result in denser fog. The fog rendering is exponential like in real life.

Depth Fog mode: The maximum intensity of the deep fog, effect will appear in the distance (relative to the camera). At `1.0` the fog will fully obscure the scene, at `0.0` the fog will not be visible.

---

float **fog_depth_begin** = `10.0`

- void **set_fog_depth_begin**(value: float)

- float **get_fog_depth_begin**()

The fog's depth starting distance from the camera. Only available when fog_mode is set to FOG_MODE_DEPTH.

---

float **fog_depth_curve** = `1.0`

- void **set_fog_depth_curve**(value: float)
- float **get_fog_depth_curve**()

The fog depth's intensity curve. A number of presets are available in the Inspector by right-clicking the curve. Only available when fog_mode is set to FOG_MODE_DEPTH.

---

float **fog_depth_end** = `100.0`

- void **set_fog_depth_end**(value: float)
- float **get_fog_depth_end**()

The fog's depth end distance from the camera. If this value is set to `0`, it will be equal to the current camera's Camera3D.far value. Only available when fog_mode is set to FOG_MODE_DEPTH.

---

bool **fog_enabled** = `false`

- void **set_fog_enabled**(value: bool)
- bool **is_fog_enabled**()

If `true`, fog effects are enabled.

---

float **fog_height** = `0.0`

- void **set_fog_height**(value: float)
- float **get_fog_height**()

The height at which the height fog effect begins.

---

float **fog_height_density** = `0.0`

- void **set_fog_height_density**(value: float)
- float **get_fog_height_density**()

The density used to increase fog as height decreases. To make fog increase as height increases, use a negative value.

Color **fog_light_color** = `Color(0.518, 0.553, 0.608, 1)`

- void **set_fog_light_color**(value: Color)
- Color **get_fog_light_color**()

The fog's color.

float **fog_light_energy** = `1.0`

- void **set_fog_light_energy**(value: float)
- float **get_fog_light_energy**()

The fog's brightness. Higher values result in brighter fog.

FogMode **fog_mode** = `0`

- void **set_fog_mode**(value: FogMode)
- FogMode **get_fog_mode**()

The fog mode. See FogMode for possible values.

float **fog_sky_affect** = `1.0`

- void **set_fog_sky_affect**(value: float)
- float **get_fog_sky_affect**()

The factor to use when affecting the sky with non-volumetric fog. `1.0` means that fog can fully obscure the sky. Lower values reduce the impact of fog on sky rendering, with `0.0` not affecting sky rendering at all.

Note: fog_sky_affect has no visual effect if fog_aerial_perspective is `1.0`.

float **fog_sun_scatter** = `0.0` 🔗

- void **set_fog_sun_scatter**(value: float)
- float **get_fog_sun_scatter**()

If set above `0.0`, renders the scene's directional light(s) in the fog color depending on the view angle. This can be used to give the impression that the sun is "piercing" through the fog.

GlowBlendMode **glow_blend_mode** = `2` 🔗

- void **set_glow_blend_mode**(value: GlowBlendMode)
- GlowBlendMode **get_glow_blend_mode**()

The glow blending mode.

Note: glow_blend_mode has no effect when using the Compatibility rendering method, due to this rendering method using a simpler glow implementation optimized for low-end devices.

---

float **glow_bloom** = `0.0` 🔗

- void **set_glow_bloom**(value: float)
- float **get_glow_bloom**()

The bloom's intensity. If set to a value higher than `0`, this will make glow visible in areas darker than the glow_hdr_threshold.

---

bool **glow_enabled** = `false` 🔗

- void **set_glow_enabled**(value: bool)
- bool **is_glow_enabled**()

If `true`, the glow effect is enabled. This simulates real world eye/camera behavior where bright pixels bleed onto surrounding pixels.

Note: When using the Mobile rendering method, glow looks different due to the lower dynamic range available in the Mobile rendering method.

Note: When using the Compatibility rendering method, glow uses a different implementation with some properties being unavailable and hidden from the inspector: `glow_levels/*`, glow_normalized, glow_strength, glow_blend_mode, glow_mix, glow_map, and glow_map_strength. This implementation is optimized to run on low-end devices and is less flexible as a result.

float **glow_hdr_luminance_cap** = `12.0`

- void **set_glow_hdr_luminance_cap**(value: float)
- float **get_glow_hdr_luminance_cap**()

The higher threshold of the HDR glow. Areas brighter than this threshold will be clamped for the purposes of the glow effect.

float **glow_hdr_scale** = `2.0`

- void **set_glow_hdr_bleed_scale**(value: float)
- float **get_glow_hdr_bleed_scale**()

The bleed scale of the HDR glow.

float **glow_hdr_threshold** = `1.0`

- void **set_glow_hdr_bleed_threshold**(value: float)
- float **get_glow_hdr_bleed_threshold**()

The lower threshold of the HDR glow. When using the Mobile rendering method (which only supports a lower dynamic range up to `2.0`), this may need to be below `1.0` for glow to be visible. A value of `0.9` works well in this case. This value also needs to be decreased below `1.0` when using glow in 2D, as 2D rendering is performed in SDR.

float **glow_intensity** = `0.8`

- void **set_glow_intensity**(value: float)
- float **get_glow_intensity**()

The overall brightness multiplier of the glow effect. When using the Mobile rendering method (which only supports a lower dynamic range up to `2.0`), this should be increased to `1.5` to compensate.

---

float **glow_levels/1** = `0.0`

- void **set_glow_level**(idx: int, intensity: float)
- float **get_glow_level**(idx: int) const

The intensity of the 1st level of glow. This is the most "local" level (least blurry).

Note: glow_levels/1 has no effect when using the Compatibility rendering method, due to this rendering method using a simpler glow implementation optimized for low-end devices.

---

float **glow_levels/2** = `0.0`

- void **set_glow_level**(idx: int, intensity: float)
- float **get_glow_level**(idx: int) const

The intensity of the 2nd level of glow.

Note: glow_levels/2 has no effect when using the Compatibility rendering method, due to this rendering method using a simpler glow implementation optimized for low-end devices.

---

float **glow_levels/3** = `1.0`

- void **set_glow_level**(idx: int, intensity: float)
- float **get_glow_level**(idx: int) const

The intensity of the 3rd level of glow.

Note: glow_levels/3 has no effect when using the Compatibility rendering method, due to this rendering method using a simpler glow implementation optimized for low-end devices.

---

float **glow_levels/4** = `0.0`

- void **set_glow_level**(idx: int, intensity: float)
- float **get_glow_level**(idx: int) const

The intensity of the 4th level of glow.

Note: glow_levels/4 has no effect when using the Compatibility rendering method, due to this rendering method using a simpler glow implementation optimized for low-end devices.

float **glow_levels/5** = `1.0`

- void **set_glow_level**(idx: int, intensity: float)
- float **get_glow_level**(idx: int) const

The intensity of the 5th level of glow.

Note: glow_levels/5 has no effect when using the Compatibility rendering method, due to this rendering method using a simpler glow implementation optimized for low-end devices.

float **glow_levels/6** = `0.0`

- void **set_glow_level**(idx: int, intensity: float)
- float **get_glow_level**(idx: int) const

The intensity of the 6th level of glow.

Note: glow_levels/6 has no effect when using the Compatibility rendering method, due to this rendering method using a simpler glow implementation optimized for low-end devices.

float **glow_levels/7** = `0.0`

- void **set_glow_level**(idx: int, intensity: float)
- float **get_glow_level**(idx: int) const

The intensity of the 7th level of glow. This is the most "global" level (blurriest).

Note: glow_levels/7 has no effect when using the Compatibility rendering method, due to this rendering method using a simpler glow implementation optimized for low-end devices.

Texture **glow_map** 🔗

- void **set_glow_map**(value: Texture)
- Texture **get_glow_map**()

The texture that should be used as a glow map to *multiply* the resulting glow color according to glow_map_strength. This can be used to create a "lens dirt" effect. The texture's RGB color channels are used for modulation, but the alpha channel is ignored.

Note: The texture will be stretched to fit the screen. Therefore, it's recommended to use a texture with an aspect ratio that matches your project's base aspect ratio (typically 16:9).

Note: glow_map has no effect when using the Compatibility rendering method, due to this rendering method using a simpler glow implementation optimized for low-end devices.

---

float **glow_map_strength** = `0.8` 🔗

- void **set_glow_map_strength**(value: float)
- float **get_glow_map_strength**()

How strong of an impact the glow_map should have on the overall glow effect. A strength of `0.0` means the glow map has no effect on the overall glow effect. A strength of `1.0` means the glow has a full effect on the overall glow effect (and can turn off glow entirely in specific areas of the screen if the glow map has black areas).

Note: glow_map_strength has no effect when using the Compatibility rendering method, due to this rendering method using a simpler glow implementation optimized for low-end devices.

---

float **glow_mix** = `0.05` 🔗

- void **set_glow_mix**(value: float)
- float **get_glow_mix**()

When using the GLOW_BLEND_MODE_MIX glow_blend_mode, this controls how much the source image is blended with the glow layer. A value of `0.0` makes the glow rendering invisible, while a value of `1.0` is equivalent to GLOW_BLEND_MODE_REPLACE.

Note: glow_mix has no effect when using the Compatibility rendering method, due to this rendering method using a simpler glow implementation optimized for low-end devices.

---

bool **glow_normalized** = `false` 🔗

- void **set_glow_normalized**(value: bool)
- bool **is_glow_normalized**()

If `true`, glow levels will be normalized so that summed together their intensities equal `1.0`.

Note: glow_normalized has no effect when using the Compatibility rendering method, due to this rendering method using a simpler glow implementation optimized for low-end devices.

---

float **glow_strength** = `1.0` 🔗

- void **set_glow_strength**(value: float)
- float **get_glow_strength**()

The strength of the glow effect. This applies as the glow is blurred across the screen and increases the distance and intensity of the blur. When using the Mobile rendering method, this should be increased to compensate for the lower dynamic range.

Note: glow_strength has no effect when using the Compatibility rendering method, due to this rendering method using a simpler glow implementation optimized for low-end devices.

ReflectionSource **reflected_light_source** = `0` 🔗

- void **set_reflection_source**(value: ReflectionSource)
- ReflectionSource **get_reflection_source**()

The reflected (specular) light source.

---

float **sdfgi_bounce_feedback** = `0.5` 🔗

- void **set_sdfgi_bounce_feedback**(value: float)
- float **get_sdfgi_bounce_feedback**()

The energy multiplier applied to light every time it bounces from a surface when using SDFGI. Values greater than `0.0` will simulate multiple bounces, resulting in a more realistic appearance. Increasing sdfgi_bounce_feedback generally has no performance impact. See also sdfgi_energy.

Note: Values greater than `0.5` can cause infinite feedback loops and should be avoided in scenes with bright materials.

Note: If sdfgi_bounce_feedback is `0.0`, indirect lighting will not be represented in reflections as light will only bounce one time.

---

float **sdfgi_cascade0_distance** = `12.8` 🔗

- void **set_sdfgi_cascade0_distance**(value: float)
- float **get_sdfgi_cascade0_distance**()

Note: This property is linked to sdfgi_min_cell_size and sdfgi_max_distance. Changing its value will automatically change those properties as well.

---

int **sdfgi_cascades** = `4` 🔗

- void **set_sdfgi_cascades**(value: int)
- int **get_sdfgi_cascades**()

The number of cascades to use for SDFGI (between 1 and 8). A higher number of cascades allows displaying SDFGI further away while preserving detail up close, at the cost of performance. When using SDFGI on small-scale levels, sdfgi_cascades can often be decreased between `1` and `4` to improve performance.

---

bool **sdfgi_enabled** = `false` 🔗

- void **set_sdfgi_enabled**(value: bool)
- bool **is_sdfgi_enabled**()

If `true`, enables signed distance field global illumination for meshes that have their GeometryInstance3D.gi_mode set to GeometryInstance3D.GI_MODE_STATIC. SDFGI is a real-time global illumination technique that works well with procedurally generated and user-built levels, including in situations where geometry is created during gameplay. The signed distance field is automatically generated around the camera as it moves. Dynamic lights are supported, but dynamic occluders and emissive surfaces are not.

Note: SDFGI is only supported in the Forward+ rendering method, not Mobile or Compatibility.

Performance: SDFGI is relatively demanding on the GPU and is not suited to low-end hardware such as integrated graphics (consider LightmapGI instead). To improve SDFGI performance, enable ProjectSettings.rendering/global_illumination/gi/use_half_resolution in the Project Settings.

Note: Meshes should have sufficiently thick walls to avoid light leaks (avoid one-sided walls). For interior levels, enclose your level geometry in a sufficiently large box and bridge the loops to close the mesh.

---

float **sdfgi_energy** = `1.0`

- void **set_sdfgi_energy**(value: float)
- float **get_sdfgi_energy**()

The energy multiplier to use for SDFGI. Higher values will result in brighter indirect lighting and reflections. See also sdfgi_bounce_feedback.

---

float **sdfgi_max_distance** = `204.8`

- void **set_sdfgi_max_distance**(value: float)
- float **get_sdfgi_max_distance**()

The maximum distance at which SDFGI is visible. Beyond this distance, environment lighting or other sources of GI such as ReflectionProbe will be used as a fallback.

Note: This property is linked to sdfgi_min_cell_size and sdfgi_cascade0_distance. Changing its value will automatically change those properties as well.

---

float **sdfgi_min_cell_size** = `0.2`

- void **set_sdfgi_min_cell_size**(value: float)
- float **get_sdfgi_min_cell_size**()

The cell size to use for the closest SDFGI cascade (in 3D units). Lower values allow SDFGI to be more precise up close, at the cost of making SDFGI updates more demanding. This can cause stuttering when the camera moves fast. Higher values allow SDFGI to cover more ground, while also reducing the performance impact of SDFGI updates.

Note: This property is linked to sdfgi_max_distance and sdfgi_cascade0_distance. Changing its value will automatically change those properties as well.

---

float **sdfgi_normal_bias** = `1.1`

- void **set_sdfgi_normal_bias**(value: float)
- float **get_sdfgi_normal_bias**()

The normal bias to use for SDFGI probes. Increasing this value can reduce visible streaking artifacts on sloped surfaces, at the cost of increased light leaking.

---

float **sdfgi_probe_bias** = `1.1`

- void **set_sdfgi_probe_bias**(value: float)
- float **get_sdfgi_probe_bias**()

The constant bias to use for SDFGI probes. Increasing this value can reduce visible streaking artifacts on sloped surfaces, at the cost of increased light leaking.

---

bool **sdfgi_read_sky_light** = `true`

- void **set_sdfgi_read_sky_light**(value: bool)
- bool **is_sdfgi_reading_sky_light**()

If `true`, SDFGI takes the environment lighting into account. This should be set to `false` for interior scenes.

---

bool **sdfgi_use_occlusion** = `false`

- void **set_sdfgi_use_occlusion**(value: bool)
- bool **is_sdfgi_using_occlusion**()

If `true`, SDFGI uses an occlusion detection approach to reduce light leaking. Occlusion may however introduce dark blotches in certain spots, which may be undesired in mostly outdoor scenes. sdfgi_use_occlusion has a performance impact and should only be enabled when needed.

SDFGIYScale **sdfgi_y_scale** = `1`

- void **set_sdfgi_y_scale**(value: SDFGIYScale)
- SDFGIYScale **get_sdfgi_y_scale**()

The Y scale to use for SDFGI cells. Lower values will result in SDFGI cells being packed together more closely on the Y axis. This is used to balance between quality and covering a lot of vertical ground. sdfgi_y_scale should be set depending on how vertical your scene is (and how fast your camera may move on the Y axis).

Sky **sky**

- void **set_sky**(value: Sky)
- Sky **get_sky**()

The Sky resource used for this **Environment**.

---

float **sky_custom_fov** = `0.0`

- void **set_sky_custom_fov**(value: float)
- float **get_sky_custom_fov**()

If set to a value greater than `0.0`, overrides the field of view to use for sky rendering. If set to `0.0`, the same FOV as the current Camera3D is used for sky rendering.

Vector3 **sky_rotation** = `Vector3(0, 0, 0)`

- void **set_sky_rotation**(value: Vector3)
- Vector3 **get_sky_rotation**()

The rotation to use for sky rendering.

---

float **ssao_ao_channel_affect** = `0.0`

- void **set_ssao_ao_channel_affect**(value: float)
- float **get_ssao_ao_channel_affect**()

The screen-space ambient occlusion intensity on materials that have an AO texture defined. Values higher than `0` will make the SSAO effect visible in areas darkened by AO textures.

---

float **ssao_detail** = `0.5`

- void **set_ssao_detail**(value: float)
- float **get_ssao_detail**()

Sets the strength of the additional level of detail for the screen-space ambient occlusion effect. A high value makes the detail pass more prominent, but it may contribute to aliasing in your final image.

---

bool **ssao_enabled** = `false`

- void **set_ssao_enabled**(value: bool)
- bool **is_ssao_enabled**()

If `true`, the screen-space ambient occlusion effect is enabled. This darkens objects' corners and cavities to simulate ambient light not reaching the entire object as in real life. This works well for small, dynamic objects, but baked lighting or ambient occlusion textures will do a better job at displaying ambient occlusion on large static objects. Godot uses a form of SSAO called Adaptive Screen Space Ambient Occlusion which is itself a form of Horizon Based Ambient Occlusion.

Note: SSAO is only supported in the Forward+ rendering method, not Mobile or Compatibility.

---

float **ssao_horizon** = `0.06`

- void **set_ssao_horizon**(value: float)
- float **get_ssao_horizon**()

The threshold for considering whether a given point on a surface is occluded or not represented as an angle from the horizon mapped into the `0.0-1.0` range. A value of `1.0` results in no occlusion.

---

float **ssao_intensity** = `2.0`

- void **set_ssao_intensity**(value: float)
- float **get_ssao_intensity**()

The primary screen-space ambient occlusion intensity. Acts as a multiplier for the screen-space ambient occlusion effect. A higher value results in darker occlusion.

---

float **ssao_light_affect** = `0.0`

- void **set_ssao_direct_light_affect**(value: float)
- float **get_ssao_direct_light_affect**()

The screen-space ambient occlusion intensity in direct light. In real life, ambient occlusion only applies to indirect light, which means its effects can't be seen in direct light. Values higher than `0` will make the SSAO effect visible in direct light.

---

float **ssao_power** = `1.5`

- void **set_ssao_power**(value: float)
- float **get_ssao_power**()

The distribution of occlusion. A higher value results in darker occlusion, similar to ssao_intensity, but with a sharper falloff.

---

float **ssao_radius** = `1.0` 🔗

- void **set_ssao_radius**(value: float)
- float **get_ssao_radius**()

The distance at which objects can occlude each other when calculating screen-space ambient occlusion. Higher values will result in occlusion over a greater distance at the cost of performance and quality.

---

float **ssao_sharpness** = `0.98` 🔗

- void **set_ssao_sharpness**(value: float)
- float **get_ssao_sharpness**()

The amount that the screen-space ambient occlusion effect is allowed to blur over the edges of objects. Setting too high will result in aliasing around the edges of objects. Setting too low will make object edges appear blurry.

---

bool **ssil_enabled** = `false` 🔗

- void **set_ssil_enabled**(value: bool)
- bool **is_ssil_enabled**()

If `true`, the screen-space indirect lighting effect is enabled. Screen space indirect lighting is a form of indirect lighting that allows diffuse light to bounce between nearby objects. Screen-space indirect lighting works very similarly to screen-space ambient occlusion, in that it only affects a limited range. It is intended to be used along with a form of proper global illumination like SDFGI or VoxelGI. Screen-space indirect lighting is not affected by individual light's Light3D.light_indirect_energy.

Note: SSIL is only supported in the Forward+ rendering method, not Mobile or Compatibility.

---

float **ssil_intensity** = `1.0` 🔗

- void **set_ssil_intensity**(value: float)
- float **get_ssil_intensity**()

The brightness multiplier for the screen-space indirect lighting effect. A higher value will result in brighter light.

---

float **ssil_normal_rejection** = `1.0` 🔗

- void **set_ssil_normal_rejection**(value: float)
- float **get_ssil_normal_rejection**()

Amount of normal rejection used when calculating screen-space indirect lighting. Normal rejection uses the normal of a given sample point to reject samples that are facing away from the current pixel. Normal rejection is necessary to avoid light leaking when only one side of an object is illuminated. However, normal rejection can be disabled if light leaking is desirable, such as when the scene mostly contains emissive objects that emit light from faces that cannot be seen from the camera.

---

float **ssil_radius** = `5.0` 🔗

- void **set_ssil_radius**(value: float)
- float **get_ssil_radius**()

The distance that bounced lighting can travel when using the screen space indirect lighting effect. A larger value will result in light bouncing further in a scene, but may result in under-sampling artifacts which look like long spikes surrounding light sources.

---

float **ssil_sharpness** = `0.98` 🔗

- void **set_ssil_sharpness**(value: float)
- float **get_ssil_sharpness**()

The amount that the screen-space indirect lighting effect is allowed to blur over the edges of objects. Setting too high will result in aliasing around the edges of objects. Setting too low will make object edges appear blurry.

---

float **ssr_depth_tolerance** = `0.2` 🔗

- void **set_ssr_depth_tolerance**(value: float)
- float **get_ssr_depth_tolerance**()

The depth tolerance for screen-space reflections.

---

bool **ssr_enabled** = `false` 🔗

- void **set_ssr_enabled**(value: bool)
- bool **is_ssr_enabled**()

If `true`, screen-space reflections are enabled. Screen-space reflections are more accurate than reflections from VoxelGIs or ReflectionProbes, but are slower and can't reflect surfaces occluded by others.

Note: SSR is only supported in the Forward+ rendering method, not Mobile or Compatibility.

---

float **ssr_fade_in** = `0.15` 🔗

- void **set_ssr_fade_in**(value: float)
- float **get_ssr_fade_in**()

The fade-in distance for screen-space reflections. Affects the area from the reflected material to the screen-space reflection. Only positive values are valid (negative values will be clamped to `0.0`).

---

float **ssr_fade_out** = `2.0` 🔗

- void **set_ssr_fade_out**(value: float)
- float **get_ssr_fade_out**()

The fade-out distance for screen-space reflections. Affects the area from the screen-space reflection to the "global" reflection. Only positive values are valid (negative values will be clamped to `0.0`).

---

int **ssr_max_steps** = `64` 🔗

- void **set_ssr_max_steps**(value: int)
- int **get_ssr_max_steps**()

The maximum number of steps for screen-space reflections. Higher values are slower.

---

float **tonemap_exposure** = `1.0` 🔗

- void **set_tonemap_exposure**(value: float)
- float **get_tonemap_exposure**()

The default exposure used for tonemapping. Higher values result in a brighter image. See also tonemap_white.

ToneMapper **tonemap_mode** = `0` 🔗

- void **set_tonemapper**(value: ToneMapper)
- ToneMapper **get_tonemapper**()

The tonemapping mode to use. Tonemapping is the process that "converts" HDR values to be suitable for rendering on an LDR display. (Godot doesn't support rendering on HDR displays yet.)

---

float **tonemap_white** = `1.0` 🔗

- void **set_tonemap_white**(value: float)
- float **get_tonemap_white**()

The white reference value for tonemapping (also called "whitepoint"). Higher values can make highlights look less blown out, and will also slightly darken the whole scene as a result. Only effective if the tonemap_mode isn't set to TONE_MAPPER_LINEAR. See also tonemap_exposure.

Color **volumetric_fog_albedo** = `Color(1, 1, 1, 1)` 🔗

- void **set_volumetric_fog_albedo**(value: Color)
- Color **get_volumetric_fog_albedo**()

The Color of the volumetric fog when interacting with lights. Mist and fog have an albedo close to `Color(1, 1, 1, 1)` while smoke has a darker albedo.

---

float **volumetric_fog_ambient_inject** = `0.0` 🔗

- void **set_volumetric_fog_ambient_inject**(value: float)

- float **get_volumetric_fog_ambient_inject**()

Scales the strength of ambient light used in the volumetric fog. A value of `0.0` means that ambient light will not impact the volumetric fog. volumetric_fog_ambient_inject has a small performance cost when set above `0.0`.

Note: This has no visible effect if volumetric_fog_density is `0.0` or if volumetric_fog_albedo is a fully black color.

---

float **volumetric_fog_anisotropy** = `0.2` 🔗

- void **set_volumetric_fog_anisotropy**(value: float)
- float **get_volumetric_fog_anisotropy**()

The direction of scattered light as it goes through the volumetric fog. A value close to `1.0` means almost all light is scattered forward. A value close to `0.0` means light is scattered equally in all directions. A value close to `-1.0` means light is scattered mostly backward. Fog and mist scatter light slightly forward, while smoke scatters light equally in all directions.

---

float **volumetric_fog_density** = `0.05` 🔗

- void **set_volumetric_fog_density**(value: float)
- float **get_volumetric_fog_density**()

The base *exponential* density of the volumetric fog. Set this to the lowest density you want to have globally. FogVolumes can be used to add to or subtract from this density in specific areas. Fog rendering is exponential as in real life.

A value of `0.0` disables global volumetric fog while allowing FogVolumes to display volumetric fog in specific areas.

To make volumetric fog work as a volumetric *lighting* solution, set volumetric_fog_density to the lowest non-zero value (`0.0001`) then increase lights' Light3D.light_volumetric_fog_energy to values between `10000` and `100000` to compensate for the very low density.

---

float **volumetric_fog_detail_spread** = `2.0` 🔗

- void **set_volumetric_fog_detail_spread**(value: float)
- float **get_volumetric_fog_detail_spread**()

The distribution of size down the length of the froxel buffer. A higher value compresses the froxels closer to the camera and places more detail closer to the camera.

Color **volumetric_fog_emission** = `Color(0, 0, 0, 1)` 🔗

- void **set_volumetric_fog_emission**(value: Color)
- Color **get_volumetric_fog_emission**()

The emitted light from the volumetric fog. Even with emission, volumetric fog will not cast light onto other surfaces. Emission is useful to establish an ambient color. As the volumetric fog effect uses single-scattering only, fog tends to need a little bit of emission to soften the harsh shadows.

---

float **volumetric_fog_emission_energy** = `1.0` 🔗

- void **set_volumetric_fog_emission_energy**(value: float)
- float **get_volumetric_fog_emission_energy**()

The brightness of the emitted light from the volumetric fog.

---

bool **volumetric_fog_enabled** = `false` 🔗

- void **set_volumetric_fog_enabled**(value: bool)
- bool **is_volumetric_fog_enabled**()

Enables the volumetric fog effect. Volumetric fog uses a screen-aligned froxel buffer to calculate accurate volumetric scattering in the short to medium range. Volumetric fog interacts with FogVolumes and lights to calculate localized and global fog. Volumetric fog uses a PBR single-scattering model based on extinction, scattering, and emission which it exposes to users as density, albedo, and emission.

Note: Volumetric fog is only supported in the Forward+ rendering method, not Mobile or Compatibility.

---

float **volumetric_fog_gi_inject** = `1.0` 🔗

- void **set_volumetric_fog_gi_inject**(value: float)
- float **get_volumetric_fog_gi_inject**()

Scales the strength of Global Illumination used in the volumetric fog's albedo color. A value of `0.0` means that Global Illumination will not impact the volumetric fog. volumetric_fog_gi_inject has a small performance cost when set above `0.0`.

Note: This has no visible effect if volumetric_fog_density is `0.0` or if volumetric_fog_albedo is a fully black color.

Note: Only VoxelGI and SDFGI (sdfgi_enabled) are taken into account when using volumetric_fog_gi_inject. Global illumination from LightmapGI, ReflectionProbe and SSIL (see ssil_enabled) will be ignored by volumetric fog.

---

float **volumetric_fog_length** = `64.0` 🔗

- void **set_volumetric_fog_length**(value: float)
- float **get_volumetric_fog_length**()

The distance over which the volumetric fog is computed. Increase to compute fog over a greater range, decrease to add more detail when a long range is not needed. For best quality fog, keep this as low as possible. See also ProjectSettings.rendering/environment/volumetric_fog/volume_depth.

---

float **volumetric_fog_sky_affect** = `1.0` 🔗

- void **set_volumetric_fog_sky_affect**(value: float)
- float **get_volumetric_fog_sky_affect**()

The factor to use when affecting the sky with volumetric fog. `1.0` means that volumetric fog can fully obscure the sky. Lower values reduce the impact of volumetric fog on sky rendering, with `0.0` not affecting sky rendering at all.

Note: volumetric_fog_sky_affect also affects FogVolumes, even if volumetric_fog_density is `0.0`. If you notice FogVolumes are disappearing when looking towards the sky, set volumetric_fog_sky_affect to `1.0`.

---

float **volumetric_fog_temporal_reprojection_amount** = `0.9`

- void **set_volumetric_fog_temporal_reprojection_amount**(value: float)
- float **get_volumetric_fog_temporal_reprojection_amount**()

The amount by which to blend the last frame with the current frame. A higher number results in smoother volumetric fog, but makes "ghosting" much worse. A lower value reduces ghosting but can result in the per-frame temporal jitter becoming visible.

---

bool **volumetric_fog_temporal_reprojection_enabled** = `true`

- void **set_volumetric_fog_temporal_reprojection_enabled**(value: bool)
- bool **is_volumetric_fog_temporal_reprojection_enabled**()

Enables temporal reprojection in the volumetric fog. Temporal reprojection blends the current frame's volumetric fog with the last frame's volumetric fog to smooth out jagged edges. The performance cost is minimal; however, it leads to moving FogVolumes and Light3Ds "ghosting" and leaving a trail behind them. When temporal reprojection is enabled, try to avoid moving FogVolumes or Light3Ds too fast. Short-lived dynamic lighting effects should have Light3D.light_volumetric_fog_energy set to `0.0` to avoid ghosting.

Method Descriptions

float **get_glow_level**(idx: int) const

Returns the intensity of the glow level `idx`.

---

void **set_glow_level**(idx: int, intensity: float)

Sets the intensity of the glow level `idx`. A value above `0.0` enables the level. Each level relies on the previous level. This means that enabling higher glow levels will slow down the glow effect rendering, even if previous levels aren't enabled.

FastNoiseLite

Inherits: Noise **<** Resource **<** RefCounted **<** Object

Generates noise using the FastNoiseLite library.

Description

This class generates noise using the FastNoiseLite library, which is a collection of several noise algorithms including Cellular, Perlin, Value, and more.

Most generated noise values are in the range of `[-1, 1]`, but not always. Some of the cellular noise algorithms return results above `1`.

Properties

CellularDistanceFunction	cellular_distance_function	`0`
float	cellular_jitter	`1.0`
CellularReturnType	cellular_return_type	`1`
float	domain_warp_amplitude	`30.0`
bool	domain_warp_enabled	`false`

float	domain_warp_fractal_gain	`0.5`
float	domain_warp_fractal_lacunarity	`6.0`
int	domain_warp_fractal_octaves	`5`
DomainWarpFractalType	domain_warp_fractal_type	`1`
float	domain_warp_frequency	`0.05`
DomainWarpType	domain_warp_type	`0`
float	fractal_gain	`0.5`
float	fractal_lacunarity	`2.0`
int	fractal_octaves	`5`
float	fractal_ping_pong_strength	`2.0`
FractalType	fractal_type	`1`
float	fractal_weighted_strength	`0.0`
float	frequency	`0.01`

NoiseType	noise_type	`1`
Vector3	offset	`Vector3(0, 0, 0)`
int	seed	`0`

Enumerations

enum **NoiseType**: 🔗

NoiseType **TYPE_VALUE** = `5`

A lattice of points are assigned random values then interpolated based on neighboring values.

NoiseType **TYPE_VALUE_CUBIC** = `4`

Similar to Value noise, but slower. Has more variance in peaks and valleys.

Cubic noise can be used to avoid certain artifacts when using value noise to create a bumpmap. In general, you should always use this mode if the value noise is being used for a heightmap or bumpmap.

NoiseType **TYPE_PERLIN** = `3`

A lattice of random gradients. Their dot products are interpolated to obtain values in between the lattices.

NoiseType **TYPE_CELLULAR** = `2`

Cellular includes both Worley noise and Voronoi diagrams which creates various regions of the same value.

NoiseType **TYPE_SIMPLEX** = `0`

As opposed to TYPE_PERLIN, gradients exist in a simplex lattice rather than a grid lattice, avoiding directional artifacts.

NoiseType **TYPE_SIMPLEX_SMOOTH** = `1`

Modified, higher quality version of TYPE_SIMPLEX, but slower.

---

enum **FractalType**:

FractalType **FRACTAL_NONE** = `0`

No fractal noise.

FractalType **FRACTAL_FBM** = `1`

Method using Fractional Brownian Motion to combine octaves into a fractal.

FractalType **FRACTAL_RIDGED** = `2`

Method of combining octaves into a fractal resulting in a "ridged" look.

FractalType **FRACTAL_PING_PONG** = `3`

Method of combining octaves into a fractal with a ping pong effect.

---

enum **CellularDistanceFunction**:

CellularDistanceFunction **DISTANCE_EUCLIDEAN** = `0`

Euclidean distance to the nearest point.

CellularDistanceFunction **DISTANCE_EUCLIDEAN_SQUARED** = `1`

Squared Euclidean distance to the nearest point.

CellularDistanceFunction **DISTANCE_MANHATTAN** = `2`

Manhattan distance (taxicab metric) to the nearest point.

CellularDistanceFunction **DISTANCE_HYBRID** = `3`

Blend of DISTANCE_EUCLIDEAN and DISTANCE_MANHATTAN to give curved cell boundaries

---

enum **CellularReturnType**: 🔗

CellularReturnType **RETURN_CELL_VALUE** = `0`

The cellular distance function will return the same value for all points within a cell.

CellularReturnType **RETURN_DISTANCE** = `1`

The cellular distance function will return a value determined by the distance to the nearest point.

CellularReturnType **RETURN_DISTANCE2** = `2`

The cellular distance function returns the distance to the second-nearest point.

CellularReturnType **RETURN_DISTANCE2_ADD** = `3`

The distance to the nearest point is added to the distance to the second-nearest point.

CellularReturnType **RETURN_DISTANCE2_SUB** = `4`

The distance to the nearest point is subtracted from the distance to the second-nearest point.

CellularReturnType **RETURN_DISTANCE2_MUL** = `5`

The distance to the nearest point is multiplied with the distance to the second-nearest point.

CellularReturnType **RETURN_DISTANCE2_DIV** = `6`

The distance to the nearest point is divided by the distance to the second-nearest point.

---

enum **DomainWarpType**: 🔗

DomainWarpType **DOMAIN_WARP_SIMPLEX** = `0`

The domain is warped using the simplex noise algorithm.

DomainWarpType **DOMAIN_WARP_SIMPLEX_REDUCED** = `1`

The domain is warped using a simplified version of the simplex noise algorithm.

DomainWarpType **DOMAIN_WARP_BASIC_GRID** = `2`

The domain is warped using a simple noise grid (not as smooth as the other methods, but more performant).

---

enum **DomainWarpFractalType**: 🔗

DomainWarpFractalType **DOMAIN_WARP_FRACTAL_NONE** = `0`

No fractal noise for warping the space.

DomainWarpFractalType
DOMAIN_WARP_FRACTAL_PROGRESSIVE = `1`

Warping the space progressively, octave for octave, resulting in a more "liquified" distortion.

DomainWarpFractalType
DOMAIN_WARP_FRACTAL_INDEPENDENT = `2`

Warping the space independently for each octave, resulting in a more chaotic distortion.

Property Descriptions

CellularDistanceFunction **cellular_distance_function** = `0` 🔗

- void **set_cellular_distance_function**(value: CellularDistanceFunction)
- CellularDistanceFunction **get_cellular_distance_function**()

Determines how the distance to the nearest/second-nearest point is computed. See CellularDistanceFunction for options.

float **cellular_jitter** = `1.0` 🔗

- void **set_cellular_jitter**(value: float)
- float **get_cellular_jitter**()

Maximum distance a point can move off of its grid position. Set to `0` for an even grid.

CellularReturnType **cellular_return_type** = `1` 🔗

- void **set_cellular_return_type**(value: CellularReturnType)

- CellularReturnType **get_cellular_return_type**()

Return type from cellular noise calculations. See CellularReturnType.

---

float **domain_warp_amplitude** = `30.0`

- void **set_domain_warp_amplitude**(value: float)
- float **get_domain_warp_amplitude**()

Sets the maximum warp distance from the origin.

---

bool **domain_warp_enabled** = `false`

- void **set_domain_warp_enabled**(value: bool)
- bool **is_domain_warp_enabled**()

If enabled, another FastNoiseLite instance is used to warp the space, resulting in a distortion of the noise.

---

float **domain_warp_fractal_gain** = `0.5`

- void **set_domain_warp_fractal_gain**(value: float)
- float **get_domain_warp_fractal_gain**()

Determines the strength of each subsequent layer of the noise which is used to warp the space.

A low value places more emphasis on the lower frequency base layers, while a high value puts more emphasis on the higher frequency layers.

---

float **domain_warp_fractal_lacunarity** = `6.0`

- void **set_domain_warp_fractal_lacunarity**(value: float)
- float **get_domain_warp_fractal_lacunarity**()

Octave lacunarity of the fractal noise which warps the space. Increasing this value results in higher octaves producing noise with finer details and a rougher appearance.

---

int **domain_warp_fractal_octaves** = `5`

- void **set_domain_warp_fractal_octaves**(value: int)
- int **get_domain_warp_fractal_octaves**()

The number of noise layers that are sampled to get the final value for the fractal noise which warps the space.

DomainWarpFractalType **domain_warp_fractal_type** = `1`

- void **set_domain_warp_fractal_type**(value: DomainWarpFractalType)
- DomainWarpFractalType **get_domain_warp_fractal_type**()

The method for combining octaves into a fractal which is used to warp the space. See DomainWarpFractalType.

---

float **domain_warp_frequency** = `0.05`

- void **set_domain_warp_frequency**(value: float)
- float **get_domain_warp_frequency**()

Frequency of the noise which warps the space. Low frequency results in smooth noise while high frequency results in rougher, more granular noise.

DomainWarpType **domain_warp_type** = `0`

- void **set_domain_warp_type**(value: DomainWarpType)
- DomainWarpType **get_domain_warp_type**()

Sets the warp algorithm. See DomainWarpType.

---

float **fractal_gain** = `0.5`

- void **set_fractal_gain**(value: float)
- float **get_fractal_gain**()

Determines the strength of each subsequent layer of noise in fractal noise.

A low value places more emphasis on the lower frequency base layers, while a high value puts more emphasis on the higher frequency layers.

---

float **fractal_lacunarity** = `2.0`

- void **set_fractal_lacunarity**(value: float)
- float **get_fractal_lacunarity**()

Frequency multiplier between subsequent octaves. Increasing this value results in higher octaves producing noise with finer details and a rougher appearance.

---

int **fractal_octaves** = `5`

- void **set_fractal_octaves**(value: int)
- int **get_fractal_octaves**()

The number of noise layers that are sampled to get the final value for fractal noise types.

---

float **fractal_ping_pong_strength** = `2.0`

- void **set_fractal_ping_pong_strength**(value: float)
- float **get_fractal_ping_pong_strength**()

Sets the strength of the fractal ping pong type.

FractalType **fractal_type** = `1` 🔗

- void **set_fractal_type**(value: FractalType)
- FractalType **get_fractal_type**()

The method for combining octaves into a fractal. See FractalType.

---

float **fractal_weighted_strength** = `0.0` 🔗

- void **set_fractal_weighted_strength**(value: float)
- float **get_fractal_weighted_strength**()

Higher weighting means higher octaves have less impact if lower octaves have a large impact.

---

float **frequency** = `0.01` 🔗

- void **set_frequency**(value: float)
- float **get_frequency**()

The frequency for all noise types. Low frequency results in smooth noise while high frequency results in rougher, more granular noise.

NoiseType **noise_type** = `1` 🔗

- void **set_noise_type**(value: NoiseType)
- NoiseType **get_noise_type**()

The noise algorithm used. See NoiseType.

Vector3 **offset** = `Vector3(0, 0, 0)` 🔗

- void **set_offset**(value: Vector3)
- Vector3 **get_offset**()

Translate the noise input coordinates by the given Vector3.

---

int **seed** = 0 🔗

- void **set_seed**(value: int)
- int **get_seed**()

The random number seed for all noise types.

FBXDocument

Experimental: This class may be changed or removed in future versions.

Inherits: GLTFDocument **<** Resource **<** RefCounted **<** Object

Handles FBX documents.

Description

The FBXDocument handles FBX documents. It provides methods to append data from buffers or files, generate scenes, and register/unregister document extensions.

When exporting FBX from Blender, use the "FBX Units Scale" option. The "FBX Units Scale" option sets the correct scale factor and avoids manual adjustments when re-importing into Blender, such as through glTF export.

FBXState

Experimental: This class may be changed or removed in future versions.

Inherits: GLTFState **<** Resource **<** RefCounted **<** Object

Description

The FBXState handles the state data imported from FBX files.

Properties

bool	allow_geometry_helper_nodes	`false`

Property Descriptions

bool **allow_geometry_helper_nodes** = `false`

- void **set_allow_geometry_helper_nodes**(value: bool)
- bool **get_allow_geometry_helper_nodes**()

If `true`, the import process used auxiliary nodes called geometry helper nodes. These nodes help preserve the pivots and transformations of the original 3D model during import.

FogMaterial

Inherits: Material < Resource < RefCounted < Object

A material that controls how volumetric fog is rendered, to be assigned to a FogVolume.

Description

A Material resource that can be used by FogVolumes to draw volumetric effects.

If you need more advanced effects, use a custom fog shader.

Properties

Color	albedo	`Color(1, 1, 1, 1)`
float	density	`1.0`
Texture3D	density_texture	
float	edge_fade	`0.1`
Color	emission	`Color(0, 0, 0, 1)`
float	height_falloff	`0.0`

Property Descriptions

Color **albedo** = `Color(1, 1, 1, 1)`

- void **set_albedo**(value: Color)
- Color **get_albedo**()

The single-scattering Color of the FogVolume. Internally, albedo is converted into single-scattering, which is additively blended with other FogVolumes and the Environment.volumetric_fog_albedo.

---

float **density** = `1.0`

- void **set_density**(value: float)
- float **get_density**()

The density of the FogVolume. Denser objects are more opaque, but may suffer from under-sampling artifacts that look like stripes. Negative values can be used to subtract fog from other FogVolumes or global volumetric fog.

Note: Due to limited precision, density values between `-0.001` and `0.001` (exclusive) act like `0.0`. This does not apply to Environment.volumetric_fog_density.

Texture3D **density_texture**

- void **set_density_texture**(value: Texture3D)
- Texture3D **get_density_texture**()

The 3D texture that is used to scale the density of the FogVolume. This can be used to vary fog density within the FogVolume with any kind of static pattern. For animated effects, consider using a custom fog shader.

---

float **edge_fade** = `0.1`

- void **set_edge_fade**(value: float)
- float **get_edge_fade**()

The hardness of the edges of the FogVolume. A higher value will result in softer edges, while a lower value will result in harder edges.

Color **emission** = `Color(0, 0, 0, 1)`

- void **set_emission**(value: Color)
- Color **get_emission**()

The Color of the light emitted by the FogVolume. Emitted light will not cast light or shadows on other objects, but can be useful for modulating the Color of the FogVolume independently from light sources.

---

float **height_falloff** = `0.0`

- void **set_height_falloff**(value: float)
- float **get_height_falloff**()

The rate by which the height-based fog decreases in density as height increases in world space. A high falloff will result in a sharp transition, while a low falloff will result in a smoother transition. A value of `0.0` results in uniform-density fog. The height threshold is determined by the height of the associated FogVolume.

Font

Inherits: Resource **<** RefCounted **<** Object

Inherited By: FontFile, FontVariation, SystemFont

Abstract base class for fonts and font variations.

Description

Abstract base class for different font types. It has methods for drawing text and font character introspection.

Properties

Array[Font]	fallbacks	`[]`

Methods

float	draw_char(canvas_item: RID, pos: Vector2, char: int, font_size: int, modulate: Color = Color(1, 1, 1, 1)) const
float	draw_char_outline(canvas_item: RID, pos: Vector2, char: int, font_size: int, size: int = -1, modulate: Color = Color(1, 1, 1, 1)) const

void	draw_multiline_string(canvas_item: RID, pos: Vector2, text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, max_lines: int = -1, modulate: Color = Color(1, 1, 1, 1), brk_flags: BitField[LineBreakFlag] = 3, justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const
void	draw_multiline_string_outline(canvas_item: RID, pos: Vector2, text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, max_lines: int = -1, size: int = 1, modulate: Color = Color(1, 1, 1, 1), brk_flags: BitField[LineBreakFlag] = 3, justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const
void	draw_string(canvas_item: RID, pos: Vector2, text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, modulate: Color = Color(1, 1, 1, 1), justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const
void	draw_string_outline(canvas_item: RID, pos: Vector2, text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, size: int = 1, modulate:

	Color = Color(1, 1, 1, 1), justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const
RID	find_variation(variation_coordinates: Dictionary, face_index: int = 0, strength: float = 0.0, transform: Transform2D = Transform2D(1, 0, 0, 1, 0, 0), spacing_top: int = 0, spacing_bottom: int = 0, spacing_space: int = 0, spacing_glyph: int = 0, baseline_offset: float = 0.0) const
float	get_ascent(font_size: int = 16) const
Vector2	get_char_size(char: int, font_size: int) const
float	get_descent(font_size: int = 16) const
int	get_face_count() const
String	get_font_name() const
int	get_font_stretch() const
BitField[FontStyle]	get_font_style() const
String	get_font_style_name() const

int	get_font_weight() const
float	get_height(font_size: int = 16) const
Vector2	get_multiline_string_size(text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, max_lines: int = -1, brk_flags: BitField[LineBreakFlag] = 3, justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const
Dictionary	get_opentype_features() const
Dictionary	get_ot_name_strings() const
Array[RID]	get_rids() const
int	get_spacing(spacing: SpacingType) const
Vector2	get_string_size(text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const
String	get_supported_chars() const

Dictionary	get_supported_feature_list() const
Dictionary	get_supported_variation_list() const
float	get_underline_position(font_size: int = 16) const
float	get_underline_thickness(font_size: int = 16) const
bool	has_char(char: int) const
bool	is_language_supported(language: String) const
bool	is_script_supported(script: String) const
void	set_cache_capacity(single_line: int, multi_line: int)

Property Descriptions

Array[Font] **fallbacks** = `[]` 🔗

- void **set_fallbacks**(value: Array[Font])
- Array[Font] **get_fallbacks**()

Array of fallback **Font**s to use as a substitute if a glyph is not found in this current **Font**.

If this array is empty in a FontVariation, the FontVariation.base_font's fallbacks are used instead.

Method Descriptions

float **draw_char**(canvas_item: RID, pos: Vector2, char: int, font_size: int, modulate: Color = Color(1, 1, 1, 1)) const

Draw a single Unicode character `char` into a canvas item using the font, at a given position, with `modulate` color. `pos` specifies the baseline, not the top. To draw from the top, *ascent* must be added to the Y axis.

Note: Do not use this function to draw strings character by character, use draw_string or TextLine instead.

---

float **draw_char_outline**(canvas_item: RID, pos: Vector2, char: int, font_size: int, size: int = -1, modulate: Color = Color(1, 1, 1, 1)) const

Draw a single Unicode character `char` outline into a canvas item using the font, at a given position, with `modulate` color and `size` outline size. `pos` specifies the baseline, not the top. To draw from the top, *ascent* must be added to the Y axis.

Note: Do not use this function to draw strings character by character, use draw_string or TextLine instead.

---

void **draw_multiline_string**(canvas_item: RID, pos: Vector2, text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, max_lines: int = -1, modulate: Color = Color(1, 1, 1, 1), brk_flags: BitField[LineBreakFlag] = 3, justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const 🔗

Breaks `text` into lines using rules specified by `brk_flags` and draws it into a canvas item using the font, at a given position, with `modulate` color, optionally clipping the width and aligning horizontally. `pos` specifies the baseline of the first line, not the top. To draw from the top, *ascent* must be added to the Y axis.

See also CanvasItem.draw_multiline_string.

---

void **draw_multiline_string_outline**(canvas_item: RID, pos: Vector2, text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, max_lines: int = -1, size: int = 1, modulate: Color = Color(1, 1, 1, 1), brk_flags: BitField[LineBreakFlag] = 3, justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const 🔗

Breaks `text` to the lines using rules specified by `brk_flags` and draws text outline into a canvas item using the font, at a given position, with `modulate` color and `size` outline size, optionally clipping the width and aligning horizontally. `pos` specifies the baseline of the first line, not the top. To draw from the top, *ascent* must be added to the Y axis.

See also CanvasItem.draw_multiline_string_outline.

---

void **draw_string**(canvas_item: RID, pos: Vector2, text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, modulate: Color = Color(1, 1, 1, 1), justification_flags:

BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const

Draw `text` into a canvas item using the font, at a given position, with `modulate` color, optionally clipping the width and aligning horizontally. `pos` specifies the baseline, not the top. To draw from the top, *ascent* must be added to the Y axis.

See also CanvasItem.draw_string.

---

void **draw_string_outline**(canvas_item: RID, pos: Vector2, text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, size: int = 1, modulate: Color = Color(1, 1, 1, 1), justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const

Draw `text` outline into a canvas item using the font, at a given position, with `modulate` color and `size` outline size, optionally clipping the width and aligning horizontally. `pos` specifies the baseline, not the top. To draw from the top, *ascent* must be added to the Y axis.

See also CanvasItem.draw_string_outline.

RID **find_variation**(variation_coordinates: Dictionary, face_index: int = 0, strength: float = 0.0, transform: Transform2D = Transform2D(1, 0, 0, 1, 0, 0), spacing_top: int = 0, spacing_bottom: int = 0, spacing_space: int = 0, spacing_glyph: int = 0, baseline_offset: float = 0.0) const

Returns TextServer RID of the font cache for specific variation.

---

float **get_ascent**(font_size: int = 16) const

Returns the average font ascent (number of pixels above the baseline).

Note: Real ascent of the string is context-dependent and can be significantly different from the value returned by this function. Use it only as rough estimate (e.g. as the ascent of empty line).

Vector2 **get_char_size**(char: int, font_size: int) const

Returns the size of a character. Does not take kerning into account.

Note: Do not use this function to calculate width of the string character by character, use get_string_size or TextLine instead. The height returned is the font height (see also get_height) and has no relation to the glyph height.

---

float **get_descent**(font_size: int = 16) const

Returns the average font descent (number of pixels below the baseline).

Note: Real descent of the string is context-dependent and can be significantly different from the value returned by this function. Use it only as rough estimate (e.g. as the descent of empty line).

---

int **get_face_count**() const

Returns number of faces in the TrueType / OpenType collection.

String **get_font_name**() const

Returns font family name.

---

int **get_font_stretch**() const 🔗

Returns font stretch amount, compared to a normal width. A percentage value between `50%` and `200%`.

BitField[FontStyle] **get_font_style**() const 🔗

Returns font style flags, see FontStyle.

String **get_font_style_name**() const 🔗

Returns font style name.

---

int **get_font_weight**() const 🔗

Returns weight (boldness) of the font. A value in the `100...999` range, normal font weight is `400`, bold font weight is `700`.

---

float **get_height**(font_size: int = 16) const 🔗

Returns the total average font height (ascent plus descent) in pixels.

Note: Real height of the string is context-dependent and can be significantly different from the value returned by this function. Use it only as rough estimate (e.g. as the height of empty line).

Vector2 **get_multiline_string_size**(text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, max_lines: int = -1, brk_flags: BitField[LineBreakFlag] = 3, justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const 🔗

Returns the size of a bounding box of a string broken into the lines, taking kerning and advance into account.

See also draw_multiline_string.

Dictionary **get_opentype_features**() const 🔗

Returns a set of OpenType feature tags. More info: OpenType feature tags [https://docs.microsoft.com/en-us/typography/opentype/spec/featuretags].

Dictionary **get_ot_name_strings**() const 🔗

Returns Dictionary with OpenType font name strings (localized font names, version, description, license information, sample text, etc.).

Array[RID] **get_rids**() const 🔗

Returns Array of valid **Font** RIDs, which can be passed to the TextServer methods.

---

int **get_spacing**(spacing: SpacingType) const 🔗

Returns the spacing for the given `type` (see SpacingType).

Vector2 **get_string_size**(text: String, alignment: HorizontalAlignment = 0, width: float = -1, font_size: int = 16, justification_flags: BitField[JustificationFlag] = 3, direction: Direction = 0, orientation: Orientation = 0) const 🔗

Returns the size of a bounding box of a single-line string, taking kerning, advance and subpixel positioning into account. See also get_multiline_string_size and draw_string.

For example, to get the string size as displayed by a single-line Label, use:

GDScript

```
var string_size =
$Label.get_theme_font("font").get_string_size($Label.text,
HORIZONTAL_ALIGNMENT_LEFT, -1,
$Label.get_theme_font_size("font_size"))
```

C#

```
Label label = GetNode<Label>("Label");
Vector2 stringSize =
label.GetThemeFont("font").GetStringSize(label.Text,
HorizontalAlignment.Left, -1,
label.GetThemeFontSize("font_size"));
```

Note: Since kerning, advance and subpixel positioning are taken into account by get_string_size, using separate get_string_size calls on substrings of a string then adding the results together will return a different result compared to using a single get_string_size call on the full string.

Note: Real height of the string is context-dependent and can be significantly different from the value returned by get_height.

String **get_supported_chars**() const 🔗

Returns a string containing all the characters available in the font.

If a given character is included in more than one font data source, it appears only once in the returned string.

Dictionary **get_supported_feature_list**() const 🔗

Returns list of OpenType features supported by font.

Dictionary **get_supported_variation_list**() const 🔗

Returns list of supported variation coordinates [https://docs.microsoft.com/en-us/typography/opentype/spec/dvaraxisreg], each coordinate is returned as `tag: Vector3i(min_value,max_value,default_value)`.

Font variations allow for continuous change of glyph characteristics along some given design axis, such as weight, width or slant.

To print available variation axes of a variable font:

```
var fv = FontVariation.new()
fv.base_font = load("res://RobotoFlex.ttf")
var variation_list = fv.get_supported_variation_list()
for tag in variation_list:
    var name =
TextServerManager.get_primary_interface().tag_to_name(tag)
    var values = variation_list[tag]
    print("variation axis: %s (%d)\n\tmin, max, default: %s" %
[name, tag, values])
```

Note: To set and get variation coordinates of a FontVariation, use FontVariation.variation_opentype.

---

float **get_underline_position**(font_size: int = 16) const 🔗

Returns average pixel offset of the underline below the baseline.

Note: Real underline position of the string is context-dependent and can be significantly different from the value returned by this function. Use it only as rough estimate.

---

float **get_underline_thickness**(font_size: int = 16) const

Returns average thickness of the underline.

Note: Real underline thickness of the string is context-dependent and can be significantly different from the value returned by this function. Use it only as rough estimate.

---

bool **has_char**(char: int) const

Returns `true` if a Unicode `char` is available in the font.

---

bool **is_language_supported**(language: String) const

Returns `true`, if font supports given language (ISO 639 [https://en.wikipedia.org/wiki/ISO_639-1] code).

---

bool **is_script_supported**(script: String) const

Returns `true`, if font supports given script (ISO 15924 [https://en.wikipedia.org/wiki/ISO_15924] code).

---

void **set_cache_capacity**(single_line: int, multi_line: int)

Sets LRU cache capacity for `draw_*` methods.

FontFile

Inherits: Font **<** Resource **<** RefCounted **<** Object

Holds font source data and prerendered glyph cache, imported from a dynamic or a bitmap font.

Description

FontFile contains a set of glyphs to represent Unicode characters imported from a font file, as well as a cache of rasterized glyphs, and a set of fallback Fonts to use.

Use FontVariation to access specific OpenType variation of the font, create simulated bold / slanted version, and draw lines of text.

For more complex text processing, use FontVariation in conjunction with TextLine or TextParagraph.

Supported font formats:

- Dynamic font importer: TrueType (.ttf), TrueType collection (.ttc), OpenType (.otf), OpenType collection (.otc), WOFF (.woff), WOFF2 (.woff2), Type 1 (.pfb, .pfm).
- Bitmap font importer: AngelCode BMFont (.fnt, .font), text and binary (version 3) format variants.
- Monospace image font importer: All supported image formats.

Note: A character is a symbol that represents an item (letter, digit etc.) in an abstract way.

Note: A glyph is a bitmap or a shape used to draw one or more characters in a context-dependent manner. Glyph indices are bound to the specific font data source.

Note: If none of the font data sources contain glyphs for a character used in a string, the character in question will be replaced with a box displaying its hexadecimal code.

GDScript

```
var f = load("res://BarlowCondensed-Bold.ttf")
$Label.add_theme_font_override("font", f)
$Label.add_theme_font_size_override("font_size", 64)
```

C#

```
var f = ResourceLoader.Load<FontFile>("res://BarlowCondensed-Bold.ttf");
GetNode("Label").AddThemeFontOverride("font", f);
GetNode("Label").AddThemeFontSizeOverride("font_size", 64);
```

Tutorials

- Runtime file loading and saving

Properties

bool	allow_system_fallback	`true`
FontAntialiasing	antialiasing	`1`
PackedByteArray	data	`PackedByteArray()`
bool	disable_embedded_bitmaps	`true`
int	fixed_size	`0`
FixedSizeScaleMode	fixed_size_scale_mode	`0`
String	font_name	`""`

int	font_stretch	`100`
BitField[FontStyle]	font_style	`0`
int	font_weight	`400`
bool	force_autohinter	`false`
bool	generate_mipmaps	`false`
Hinting	hinting	`1`
int	msdf_pixel_range	`16`
int	msdf_size	`48`
bool	multichannel_signed_distance_field	`false`
Dictionary	opentype_feature_overrides	`{}`
float	oversampling	`0.0`
String	style_name	`""`
SubpixelPositioning	subpixel_positioning	`1`

Methods

void	clear_cache()
void	clear_glyphs(cache_index: int, size: Vector2i)
void	clear_kerning_map(cache_index: int, size: int)
void	clear_size_cache(cache_index: int)
void	clear_textures(cache_index: int, size: Vector2i)
float	get_cache_ascent(cache_index: int, size: int) const
int	get_cache_count() const
float	get_cache_descent(cache_index: int, size: int) const
float	get_cache_scale(cache_index: int, size: int) const
float	get_cache_underline_position(cache_index: int, size: int) const
float	get_cache_underline_thickness(cache_index: int, size: int) const
int	get_char_from_glyph_index(size: int, glyph_index: int) const
float	get_embolden(cache_index: int) const

float	get_extra_baseline_offset(cache_index: int) const
int	get_extra_spacing(cache_index: int, spacing: SpacingType) const
int	get_face_index(cache_index: int) const
Vector2	get_glyph_advance(cache_index: int, size: int, glyph: int) const
int	get_glyph_index(size: int, char: int, variation_selector: int) const
PackedInt32Array	get_glyph_list(cache_index: int, size: Vector2i) const
Vector2	get_glyph_offset(cache_index: int, size: Vector2i, glyph: int) const
Vector2	get_glyph_size(cache_index: int, size: Vector2i, glyph: int) const
int	get_glyph_texture_idx(cache_index: int, size: Vector2i, glyph: int) const
Rect2	get_glyph_uv_rect(cache_index: int, size: Vector2i, glyph: int) const
Vector2	get_kerning(cache_index: int, size: int, glyph_pair: Vector2i) const

Array[Vector2i]	get_kerning_list(cache_index: int, size: int) const
bool	get_language_support_override(language: String) const
PackedStringArray	get_language_support_overrides() const
bool	get_script_support_override(script: String) const
PackedStringArray	get_script_support_overrides() const
Array[Vector2i]	get_size_cache_list(cache_index: int) const
int	get_texture_count(cache_index: int, size: Vector2i) const
Image	get_texture_image(cache_index: int, size: Vector2i, texture_index: int) const
PackedInt32Array	get_texture_offsets(cache_index: int, size: Vector2i, texture_index: int) const
Transform2D	get_transform(cache_index: int) const
Dictionary	get_variation_coordinates(cache_index: int) const
Error	load_bitmap_font(path: String)
Error	load_dynamic_font(path: String)

void	remove_cache(cache_index: int)
void	remove_glyph(cache_index: int, size: Vector2i, glyph: int)
void	remove_kerning(cache_index: int, size: int, glyph_pair: Vector2i)
void	remove_language_support_override(language: String)
void	remove_script_support_override(script: String)
void	remove_size_cache(cache_index: int, size: Vector2i)
void	remove_texture(cache_index: int, size: Vector2i, texture_index: int)
void	render_glyph(cache_index: int, size: Vector2i, index: int)
void	render_range(cache_index: int, size: Vector2i, start: int, end: int)
void	set_cache_ascent(cache_index: int, size: int, ascent: float)
void	set_cache_descent(cache_index: int, size: int, descent: float)
void	set_cache_scale(cache_index: int, size: int, scale: float)

void	set_cache_underline_position(cache_index: int, size: int, underline_position: float)
void	set_cache_underline_thickness(cache_index: int, size: int, underline_thickness: float)
void	set_embolden(cache_index: int, strength: float)
void	set_extra_baseline_offset(cache_index: int, baseline_offset: float)
void	set_extra_spacing(cache_index: int, spacing: SpacingType, value: int)
void	set_face_index(cache_index: int, face_index: int)
void	set_glyph_advance(cache_index: int, size: int, glyph: int, advance: Vector2)
void	set_glyph_offset(cache_index: int, size: Vector2i, glyph: int, offset: Vector2)
void	set_glyph_size(cache_index: int, size: Vector2i, glyph: int, gl_size: Vector2)
void	set_glyph_texture_idx(cache_index: int, size: Vector2i, glyph: int, texture_idx: int)
void	set_glyph_uv_rect(cache_index: int, size: Vector2i, glyph: int, uv_rect: Rect2)

void	set_kerning(cache_index: int, size: int, glyph_pair: Vector2i, kerning: Vector2)
void	set_language_support_override(language: String, supported: bool)
void	set_script_support_override(script: String, supported: bool)
void	set_texture_image(cache_index: int, size: Vector2i, texture_index: int, image: Image)
void	set_texture_offsets(cache_index: int, size: Vector2i, texture_index: int, offset: PackedInt32Array)
void	set_transform(cache_index: int, transform: Transform2D)
void	set_variation_coordinates(cache_index: int, variation_coordinates: Dictionary)

Property Descriptions

bool **allow_system_fallback** = `true` 🔗

- void **set_allow_system_fallback**(value: bool)
- bool **is_allow_system_fallback**()

If set to `true`, system fonts can be automatically used as fallbacks.

FontAntialiasing **antialiasing** = `1` 🔗

- void **set_antialiasing**(value: FontAntialiasing)
- FontAntialiasing **get_antialiasing**()

Font anti-aliasing mode.

PackedByteArray **data** = `PackedByteArray()` 🔗

- void **set_data**(value: PackedByteArray)
- PackedByteArray **get_data**()

Contents of the dynamic font source file.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedByteArray for more details.

---

bool **disable_embedded_bitmaps** = `true` 🔗

- void **set_disable_embedded_bitmaps**(value: bool)
- bool **get_disable_embedded_bitmaps**()

If set to `true`, embedded font bitmap loading is disabled (bitmap-only and color fonts ignore this property).

---

int **fixed_size** = `0` 🔗

- void **set_fixed_size**(value: int)
- int **get_fixed_size**()

Font size, used only for the bitmap fonts.

FixedSizeScaleMode **fixed_size_scale_mode** = `0` 🔗

- void **set_fixed_size_scale_mode**(value: FixedSizeScaleMode)
- FixedSizeScaleMode **get_fixed_size_scale_mode**()

Scaling mode, used only for the bitmap fonts with fixed_size greater than zero.

String **font_name** = `""`

- void **set_font_name**(value: String)
- String **get_font_name**()

Font family name.

---

int **font_stretch** = `100`

- void **set_font_stretch**(value: int)
- int **get_font_stretch**()

Font stretch amount, compared to a normal width. A percentage value between `50%` and `200%`.

BitField[FontStyle] **font_style** = `0`

- void **set_font_style**(value: BitField[FontStyle])
- BitField[FontStyle] **get_font_style**()

Font style flags, see FontStyle.

---

int **font_weight** = `400`

- void **set_font_weight**(value: int)
- int **get_font_weight**()

Weight (boldness) of the font. A value in the `100...999` range, normal font weight is `400`, bold font weight is `700`.

---

bool **force_autohinter** = `false`

- void **set_force_autohinter**(value: bool)
- bool **is_force_autohinter**()

If set to `true`, auto-hinting is supported and preferred over font built-in hinting. Used by dynamic fonts only (MSDF fonts don't support hinting).

bool **generate_mipmaps** = `false`

- void **set_generate_mipmaps**(value: bool)
- bool **get_generate_mipmaps**()

If set to `true`, generate mipmaps for the font textures.

Hinting **hinting** = `1`

- void **set_hinting**(value: Hinting)
- Hinting **get_hinting**()

Font hinting mode. Used by dynamic fonts only.

int **msdf_pixel_range** = `16`

- void **set_msdf_pixel_range**(value: int)
- int **get_msdf_pixel_range**()

The width of the range around the shape between the minimum and maximum representable signed distance. If using font outlines, msdf_pixel_range must be set to at least *twice* the size of the largest font outline. The default msdf_pixel_range value of `16` allows outline sizes up to `8` to look correct.

int **msdf_size** = `48`

- void **set_msdf_size**(value: int)
- int **get_msdf_size**()

Source font size used to generate MSDF textures. Higher values allow for more precision, but are slower to render and require more memory. Only increase this value if you notice a visible lack of precision in glyph rendering.

bool **multichannel_signed_distance_field** = `false`

- void **set_multichannel_signed_distance_field**(value: bool)
- bool **is_multichannel_signed_distance_field**()

If set to `true`, glyphs of all sizes are rendered using single multichannel signed distance field (MSDF) generated from the dynamic font vector data. Since this approach does not rely on rasterizing the font every time its size changes, this allows for resizing the font in real-time without any performance penalty. Text will also not look grainy for Controls that are scaled down (or for Label3Ds viewed from a long distance). As a downside, font hinting is not available with MSDF. The lack of font hinting may result in less crisp and less readable fonts at small sizes.

Note: If using font outlines, msdf_pixel_range must be set to at least *twice* the size of the largest font outline.

Note: MSDF font rendering does not render glyphs with overlapping shapes correctly. Overlapping shapes are not valid per the OpenType standard, but are still commonly found in many font files, especially those converted by Google Fonts. To avoid issues with overlapping glyphs, consider downloading the font file directly from the type foundry instead of relying on Google Fonts.

Dictionary **opentype_feature_overrides** = `{}`

- void **set_opentype_feature_overrides**(value: Dictionary)
- Dictionary **get_opentype_feature_overrides**()

Font OpenType feature set override.

---

float **oversampling** = `0.0`

- void **set_oversampling**(value: float)
- float **get_oversampling**()

Font oversampling factor. If set to `0.0`, the global oversampling factor is used instead. Used by dynamic fonts only (MSDF fonts ignore oversampling).

String **style_name** = `""`

- void **set_font_style_name**(value: String)
- String **get_font_style_name**()

Font style name.

SubpixelPositioning **subpixel_positioning** = `1`

- void **set_subpixel_positioning**(value: SubpixelPositioning)
- SubpixelPositioning **get_subpixel_positioning**()

Font glyph subpixel positioning mode. Subpixel positioning provides shaper text and better kerning for smaller font sizes, at the cost of higher memory usage and lower font rasterization speed. Use TextServer.SUBPIXEL_POSITIONING_AUTO to automatically enable it based on the font size.

Method Descriptions

void **clear_cache**()

Removes all font cache entries.

---

void **clear_glyphs**(cache_index: int, size: Vector2i)

Removes all rendered glyph information from the cache entry.

Note: This function will not remove textures associated with the glyphs, use remove_texture to remove them manually.

---

void **clear_kerning_map**(cache_index: int, size: int)

Removes all kerning overrides.

---

void **clear_size_cache**(cache_index: int)

Removes all font sizes from the cache entry

---

void **clear_textures**(cache_index: int, size: Vector2i)

Removes all textures from font cache entry.

Note: This function will not remove glyphs associated with the texture, use remove_glyph to remove them manually.

---

float **get_cache_ascent**(cache_index: int, size: int) const

Returns the font ascent (number of pixels above the baseline).

---

int **get_cache_count**() const

Returns number of the font cache entries.

---

float **get_cache_descent**(cache_index: int, size: int) const

Returns the font descent (number of pixels below the baseline).

---

float **get_cache_scale**(cache_index: int, size: int) const

Returns scaling factor of the color bitmap font.

---

float **get_cache_underline_position**(cache_index: int, size: int) const

Returns pixel offset of the underline below the baseline.

---

float **get_cache_underline_thickness**(cache_index: int, size: int) const

Returns thickness of the underline in pixels.

---

int **get_char_from_glyph_index**(size: int, glyph_index: int) const 🔗

Returns character code associated with `glyph_index`, or `0` if `glyph_index` is invalid. See get_glyph_index.

---

float **get_embolden**(cache_index: int) const 🔗

Returns embolden strength, if is not equal to zero, emboldens the font outlines. Negative values reduce the outline thickness.

---

float **get_extra_baseline_offset**(cache_index: int) const 🔗

Returns extra baseline offset (as a fraction of font height).

---

int **get_extra_spacing**(cache_index: int, spacing: SpacingType) const 🔗

Returns spacing for `spacing` (see SpacingType) in pixels (not relative to the font size).

---

int **get_face_index**(cache_index: int) const 🔗

Returns an active face index in the TrueType / OpenType collection.

Vector2 **get_glyph_advance**(cache_index: int, size: int, glyph: int) const 🔗

Returns glyph advance (offset of the next glyph).

Note: Advance for glyphs outlines is the same as the base glyph advance and is not saved.

---

int **get_glyph_index**(size: int, char: int, variation_selector: int) const 🔗

Returns the glyph index of a `char`, optionally modified by the `variation_selector`.

PackedInt32Array **get_glyph_list**(cache_index: int, size: Vector2i) const 🔗

Returns list of rendered glyphs in the cache entry.

Vector2 **get_glyph_offset**(cache_index: int, size: Vector2i, glyph: int) const 🔗

Returns glyph offset from the baseline.

Vector2 **get_glyph_size**(cache_index: int, size: Vector2i, glyph: int) const 🔗

Returns glyph size.

---

int **get_glyph_texture_idx**(cache_index: int, size: Vector2i, glyph: int) const 🔗

Returns index of the cache texture containing the glyph.

Rect2 **get_glyph_uv_rect**(cache_index: int, size: Vector2i, glyph: int) const 🔗

Returns rectangle in the cache texture containing the glyph.

Vector2 **get_kerning**(cache_index: int, size: int, glyph_pair: Vector2i) const 🔗

Returns kerning for the pair of glyphs.

Array[Vector2i] **get_kerning_list**(cache_index: int, size: int) const 🔗

Returns list of the kerning overrides.

bool **get_language_support_override**(language: String) const 🔗

Returns `true` if support override is enabled for the `language`.

PackedStringArray **get_language_support_overrides**() const 🔗

Returns list of language support overrides.

bool **get_script_support_override**(script: String) const 🔗

Returns `true` if support override is enabled for the `script`.

PackedStringArray **get_script_support_overrides**() const 🔗

Returns list of script support overrides.

Array[Vector2i] **get_size_cache_list**(cache_index: int) const 🔗

Returns list of the font sizes in the cache. Each size is Vector2i with font size and outline size.

int **get_texture_count**(cache_index: int, size: Vector2i) const 🔗

Returns number of textures used by font cache entry.

Image **get_texture_image**(cache_index: int, size: Vector2i, texture_index: int) const 🔗

Returns a copy of the font cache texture image.

PackedInt32Array **get_texture_offsets**(cache_index: int, size: Vector2i, texture_index: int) const 🔗

Returns a copy of the array containing glyph packing data.

Transform2D **get_transform**(cache_index: int) const 🔗

Returns 2D transform, applied to the font outlines, can be used for slanting, flipping and rotating glyphs.

Dictionary **get_variation_coordinates**(cache_index: int) const 🔗

Returns variation coordinates for the specified font cache entry. See Font.get_supported_variation_list for more info.

Error **load_bitmap_font**(path: String) 🔗

Loads an AngelCode BMFont (.fnt, .font) bitmap font from file `path`.

Warning: This method should only be used in the editor or in cases when you need to load external fonts at run-time, such as fonts located at the `user://` directory.

Error **load_dynamic_font**(path: String) 🔗

Loads a TrueType (.ttf), OpenType (.otf), WOFF (.woff), WOFF2 (.woff2) or Type 1 (.pfb, .pfm) dynamic font from file `path`.

Warning: This method should only be used in the editor or in cases when you need to load external fonts at run-time, such as fonts located at the `user://` directory.

---

void **remove_cache**(cache_index: int) 🔗

Removes specified font cache entry.

---

void **remove_glyph**(cache_index: int, size: Vector2i, glyph: int) 🔗

Removes specified rendered glyph information from the cache entry.

Note: This function will not remove textures associated with the glyphs, use remove_texture to remove them manually.

---

void **remove_kerning**(cache_index: int, size: int, glyph_pair: Vector2i) 🔗

Removes kerning override for the pair of glyphs.

---

void **remove_language_support_override**(language: String) 🔗

Remove language support override.

---

void **remove_script_support_override**(script: String) 🔗

Removes script support override.

---

void **remove_size_cache**(cache_index: int, size: Vector2i) 🔗

Removes specified font size from the cache entry.

---

void **remove_texture**(cache_index: int, size: Vector2i, texture_index: int) 🔗

Removes specified texture from the cache entry.

Note: This function will not remove glyphs associated with the texture. Remove them manually using remove_glyph.

---

void **render_glyph**(cache_index: int, size: Vector2i, index: int) 🔗

Renders specified glyph to the font cache texture.

---

void **render_range**(cache_index: int, size: Vector2i, start: int, end: int) 🔗

Renders the range of characters to the font cache texture.

void **set_cache_ascent**(cache_index: int, size: int, ascent: float) 🔗

Sets the font ascent (number of pixels above the baseline).

void **set_cache_descent**(cache_index: int, size: int, descent: float) 🔗

Sets the font descent (number of pixels below the baseline).

void **set_cache_scale**(cache_index: int, size: int, scale: float) 🔗

Sets scaling factor of the color bitmap font.

void **set_cache_underline_position**(cache_index: int, size: int, underline_position: float) 🔗

Sets pixel offset of the underline below the baseline.

void **set_cache_underline_thickness**(cache_index: int, size: int, underline_thickness: float) 🔗

Sets thickness of the underline in pixels.

void **set_embolden**(cache_index: int, strength: float) 🔗

Sets embolden strength, if is not equal to zero, emboldens the font outlines. Negative values reduce the outline thickness.

void **set_extra_baseline_offset**(cache_index: int, baseline_offset: float) 🔗

Sets extra baseline offset (as a fraction of font height).

---

void **set_extra_spacing**(cache_index: int, spacing: SpacingType, value: int)

Sets the spacing for `spacing` (see SpacingType) to `value` in pixels (not relative to the font size).

---

void **set_face_index**(cache_index: int, face_index: int)

Sets an active face index in the TrueType / OpenType collection.

---

void **set_glyph_advance**(cache_index: int, size: int, glyph: int, advance: Vector2)

Sets glyph advance (offset of the next glyph).

Note: Advance for glyphs outlines is the same as the base glyph advance and is not saved.

---

void **set_glyph_offset**(cache_index: int, size: Vector2i, glyph: int, offset: Vector2)

Sets glyph offset from the baseline.

---

void **set_glyph_size**(cache_index: int, size: Vector2i, glyph: int, gl_size: Vector2)

Sets glyph size.

---

void **set_glyph_texture_idx**(cache_index: int, size: Vector2i, glyph: int, texture_idx: int)

Sets index of the cache texture containing the glyph.

---

void **set_glyph_uv_rect**(cache_index: int, size: Vector2i, glyph: int, uv_rect: Rect2) 🔗

Sets rectangle in the cache texture containing the glyph.

void **set_kerning**(cache_index: int, size: int, glyph_pair: Vector2i, kerning: Vector2) 🔗

Sets kerning for the pair of glyphs.

void **set_language_support_override**(language: String, supported: bool) 🔗

Adds override for Font.is_language_supported.

void **set_script_support_override**(script: String, supported: bool) 🔗

Adds override for Font.is_script_supported.

void **set_texture_image**(cache_index: int, size: Vector2i, texture_index: int, image: Image) 🔗

Sets font cache texture image.

void **set_texture_offsets**(cache_index: int, size: Vector2i, texture_index: int, offset: PackedInt32Array) 🔗

Sets array containing glyph packing data.

void **set_transform**(cache_index: int, transform: Transform2D) 🔗

Sets 2D transform, applied to the font outlines, can be used for slanting, flipping, and rotating glyphs.

void **set_variation_coordinates**(cache_index: int, variation_coordinates: Dictionary) 🔗

Sets variation coordinates for the specified font cache entry. See Font.get_supported_variation_list for more info.

FontVariation

Inherits: [Font]() **<** [Resource]() **<** [RefCounted]() **<** [Object]()

A variation of a font with additional settings.

Description

Provides OpenType variations, simulated bold / slant, and additional font settings like OpenType features and extra spacing.

To use simulated bold font variant:

GDScript

```
var fv = FontVariation.new()
fv.base_font = load("res://BarlowCondensed-Regular.ttf")
fv.variation_embolden = 1.2
$Label.add_theme_font_override("font", fv)
$Label.add_theme_font_size_override("font_size", 64)
```

C#

```
var fv = new FontVariation();
fv.SetBaseFont(ResourceLoader.Load<FontFile>
("res://BarlowCondensed-Regular.ttf"));
fv.SetVariationEmbolden(1.2);
GetNode("Label").AddThemeFontOverride("font", fv);
GetNode("Label").AddThemeFontSizeOverride("font_size", 64);
```

To set the coordinate of multiple variation axes:

```
var fv = FontVariation.new();
var ts = TextServerManager.get_primary_interface()
fv.base_font = load("res://BarlowCondensed-Regular.ttf")
fv.variation_opentype = { ts.name_to_tag("wght"): 900,
ts.name_to_tag("custom_hght"): 900 }
```

Properties

Font	base_font	
float	baseline_offset	`0.0`
Dictionary	opentype_features	`{}`
int	spacing_bottom	`0`
int	spacing_glyph	`0`
int	spacing_space	`0`
int	spacing_top	`0`
float	variation_embolden	`0.0`
int	variation_face_index	`0`
Dictionary	variation_opentype	`{}`

Transform2D	variation_transform	`Transform2D(1, 0, 0, 1, 0, 0)`

Methods

void	set_spacing(spacing: SpacingType, value: int)

Property Descriptions

Font **base_font** 🔗

- void **set_base_font**(value: Font)
- Font **get_base_font**()

Base font used to create a variation. If not set, default Theme font is used.

float **baseline_offset** = `0.0` 🔗

- void **set_baseline_offset**(value: float)
- float **get_baseline_offset**()

Extra baseline offset (as a fraction of font height).

Dictionary **opentype_features** = `{}` 🔗

- void **set_opentype_features**(value: Dictionary)
- Dictionary **get_opentype_features**()

A set of OpenType feature tags. More info: OpenType feature tags [https://docs.microsoft.com/en-us/typography/opentype/spec/featuretags].

---

int **spacing_bottom** = `0`

- void **set_spacing**(spacing: SpacingType, value: int)
- int **get_spacing**()

Extra spacing at the bottom of the line in pixels.

---

int **spacing_glyph** = `0`

- void **set_spacing**(spacing: SpacingType, value: int)
- int **get_spacing**()

Extra spacing between graphical glyphs.

---

int **spacing_space** = `0`

- void **set_spacing**(spacing: SpacingType, value: int)
- int **get_spacing**()

Extra width of the space glyphs.

---

int **spacing_top** = `0`

- void **set_spacing**(spacing: SpacingType, value: int)
- int **get_spacing**()

Extra spacing at the top of the line in pixels.

---

float **variation_embolden** = `0.0`

- void **set_variation_embolden**(value: float)
- float **get_variation_embolden**()

If is not equal to zero, emboldens the font outlines. Negative values reduce the outline thickness.

Note: Emboldened fonts might have self-intersecting outlines, which will prevent MSDF fonts and TextMesh from working correctly.

---

int **variation_face_index** = `0` 🔗

- void **set_variation_face_index**(value: int)
- int **get_variation_face_index**()

Active face index in the TrueType / OpenType collection file.

Dictionary **variation_opentype** = `{}` 🔗

- void **set_variation_opentype**(value: Dictionary)
- Dictionary **get_variation_opentype**()

Font OpenType variation coordinates. More info: OpenType variation tags [https://docs.microsoft.com/en-us/typography/opentype/spec/dvaraxisreg].

Note: This Dictionary uses OpenType tags as keys. Variation axes can be identified both by tags (int, e.g. `0x77678674`) and names (String, e.g. `wght`). Some axes might be accessible by multiple names. For example, `wght` refers to the same axis as `weight`. Tags on the other hand are unique. To convert between names and tags, use TextServer.name_to_tag and TextServer.tag_to_name.

Note: To get available variation axes of a font, use Font.get_supported_variation_list.

Transform2D **variation_transform** = `Transform2D(1, 0, 0, 1, 0, 0)`

- void **set_variation_transform**(value: Transform2D)
- Transform2D **get_variation_transform**()

2D transform, applied to the font outlines, can be used for slanting, flipping and rotating glyphs.

For example, to simulate italic typeface by slanting, apply the following transform `Transform2D(1.0, slant, 0.0, 1.0, 0.0, 0.0)`.

Method Descriptions

void **set_spacing**(spacing: SpacingType, value: int)

Sets the spacing for `spacing` (see SpacingType) to `value` in pixels (not relative to the font size).

GDExtension

Inherits: Resource **<** RefCounted **<** Object

A native library for GDExtension.

Description

The **GDExtension** resource type represents a shared library [https://en.wikipedia.org/wiki/Shared_library] which can expand the functionality of the engine. The GDExtensionManager singleton is responsible for loading, reloading, and unloading **GDExtension** resources.

Note: GDExtension itself is not a scripting language and has no relation to GDScript resources.

Tutorials

- GDExtension overview
- GDExtension example in C++

Methods

InitializationLevel	get_minimum_library_initialization_level() const
bool	is_library_open() const

Enumerations

enum **InitializationLevel**: 🔗

InitializationLevel **INITIALIZATION_LEVEL_CORE** = `0`

The library is initialized at the same time as the core features of the engine.

InitializationLevel **INITIALIZATION_LEVEL_SERVERS** = `1`

The library is initialized at the same time as the engine's servers (such as RenderingServer or PhysicsServer3D).

InitializationLevel **INITIALIZATION_LEVEL_SCENE** = `2`

The library is initialized at the same time as the engine's scene-related classes.

InitializationLevel **INITIALIZATION_LEVEL_EDITOR** = `3`

The library is initialized at the same time as the engine's editor classes. Only happens when loading the GDExtension in the editor.

Method Descriptions

InitializationLevel **get_minimum_library_initialization_level**() const 🔗

Returns the lowest level required for this extension to be properly initialized (see the InitializationLevel enum).

---

bool **is_library_open**() const 🔗

Returns `true` if this extension's library has been opened.

GDScript

Inherits: Script **<** Resource **<** RefCounted **<** Object

A script implemented in the GDScript programming language.

Description

A script implemented in the GDScript programming language, saved with the `.gd` extension. The script extends the functionality of all objects that instantiate it.

Calling new creates a new instance of the script. Object.set_script extends an existing object, if that object's class matches one of the script's base classes.

If you are looking for GDScript's built-in functions, see @GDScript instead.

Tutorials

- GDScript documentation index

Methods

Variant	new(...) vararg

Method Descriptions

Variant **new**(...) vararg 🔗

Returns a new instance of the script.

For example:

```
var MyClass = load("myclass.gd")
var instance = MyClass.new()
assert(instance.get_script() == MyClass)
```

GLTFAccessor

Inherits: Resource **<** RefCounted **<** Object

Represents a GLTF accessor.

Description

GLTFAccessor is a data structure representing GLTF a `accessor` that would be found in the `"accessors"` array. A buffer is a blob of binary data. A buffer view is a slice of a buffer. An accessor is a typed interpretation of the data in a buffer view.

Most custom data stored in GLTF does not need accessors, only buffer views (see GLTFBufferView). Accessors are for more advanced use cases such as interleaved mesh data encoded for the GPU.

Tutorials

- Buffers, BufferViews, and Accessors in Khronos glTF specification [https://github.com/KhronosGroup/glTF-Tutorials/blob/master/gltfTutorial/gltfTutorial_005_BuffersBufferViewsAccessors.md]
- Runtime file loading and saving

Properties

GLTFAccessorType	accessor_type	`0`
int	buffer_view	`-1`
int	byte_offset	`0`

int	component_type	`0`
int	count	`0`
PackedFloat64Array	max	`PackedFloat64Array()`
PackedFloat64Array	min	`PackedFloat64Array()`
bool	normalized	`false`
int	sparse_count	`0`
int	sparse_indices_buffer_view	`0`
int	sparse_indices_byte_offset	`0`
int	sparse_indices_component_type	`0`
int	sparse_values_buffer_view	`0`
int	sparse_values_byte_offset	`0`
int	type	

Enumerations

enum **GLTFAccessorType**: 🔗

GLTFAccessorType **TYPE_SCALAR** = `0`

Accessor type "SCALAR". For the glTF object model, this can be used to map to a single float, int, or bool value, or a float array.

GLTFAccessorType **TYPE_VEC2** = `1`

Accessor type "VEC2". For the glTF object model, this maps to "float2", represented in the glTF JSON as an array of two floats.

GLTFAccessorType **TYPE_VEC3** = `2`

Accessor type "VEC3". For the glTF object model, this maps to "float3", represented in the glTF JSON as an array of three floats.

GLTFAccessorType **TYPE_VEC4** = `3`

Accessor type "VEC4". For the glTF object model, this maps to "float4", represented in the glTF JSON as an array of four floats.

GLTFAccessorType **TYPE_MAT2** = `4`

Accessor type "MAT2". For the glTF object model, this maps to "float2x2", represented in the glTF JSON as an array of four floats.

GLTFAccessorType **TYPE_MAT3** = `5`

Accessor type "MAT3". For the glTF object model, this maps to "float3x3", represented in the glTF JSON as an array of nine floats.

GLTFAccessorType **TYPE_MAT4** = `6`

Accessor type "MAT4". For the glTF object model, this maps to "float4x4", represented in the glTF JSON as an array of sixteen floats.

Property Descriptions

GLTFAccessorType **accessor_type** = `0` 🔗

- void **set_accessor_type**(value: GLTFAccessorType)

- GLTFAccessorType **get_accessor_type**()

The GLTF accessor type as an enum. Possible values are 0 for "SCALAR", 1 for "VEC2", 2 for "VEC3", 3 for "VEC4", 4 for "MAT2", 5 for "MAT3", and 6 for "MAT4".

---

int **buffer_view** = `-1`

- void **set_buffer_view**(value: int)
- int **get_buffer_view**()

The index of the buffer view this accessor is referencing. If `-1`, this accessor is not referencing any buffer view.

---

int **byte_offset** = `0`

- void **set_byte_offset**(value: int)
- int **get_byte_offset**()

The offset relative to the start of the buffer view in bytes.

---

int **component_type** = `0`

- void **set_component_type**(value: int)
- int **get_component_type**()

The GLTF component type as an enum. Possible values are 5120 for "BYTE", 5121 for "UNSIGNED_BYTE", 5122 for "SHORT", 5123 for "UNSIGNED_SHORT", 5125 for "UNSIGNED_INT", and 5126 for "FLOAT". A value of 5125 or "UNSIGNED_INT" must not be used for any accessor that is not referenced by mesh.primitive.indices.

---

int **count** = `0`

- void **set_count**(value: int)
- int **get_count**()

The number of elements referenced by this accessor.

PackedFloat64Array **max** = `PackedFloat64Array()` 🔗

- void **set_max**(value: PackedFloat64Array)
- PackedFloat64Array **get_max**()

Maximum value of each component in this accessor.

PackedFloat64Array **min** = `PackedFloat64Array()` 🔗

- void **set_min**(value: PackedFloat64Array)
- PackedFloat64Array **get_min**()

Minimum value of each component in this accessor.

---

bool **normalized** = `false` 🔗

- void **set_normalized**(value: bool)
- bool **get_normalized**()

Specifies whether integer data values are normalized before usage.

---

int **sparse_count** = `0` 🔗

- void **set_sparse_count**(value: int)
- int **get_sparse_count**()

Number of deviating accessor values stored in the sparse array.

---

int **sparse_indices_buffer_view** = `0` 🔗

- void **set_sparse_indices_buffer_view**(value: int)
- int **get_sparse_indices_buffer_view**()

The index of the buffer view with sparse indices. The referenced buffer view MUST NOT have its target or byteStride properties defined. The buffer view and the optional byteOffset MUST be aligned to the componentType byte length.

---

int **sparse_indices_byte_offset** = 0

- void **set_sparse_indices_byte_offset**(value: int)
- int **get_sparse_indices_byte_offset**()

The offset relative to the start of the buffer view in bytes.

---

int **sparse_indices_component_type** = 0

- void **set_sparse_indices_component_type**(value: int)
- int **get_sparse_indices_component_type**()

The indices component data type as an enum. Possible values are 5121 for "UNSIGNED_BYTE", 5123 for "UNSIGNED_SHORT", and 5125 for "UNSIGNED_INT".

---

int **sparse_values_buffer_view** = 0

- void **set_sparse_values_buffer_view**(value: int)
- int **get_sparse_values_buffer_view**()

The index of the bufferView with sparse values. The referenced buffer view MUST NOT have its target or byteStride properties defined.

---

int **sparse_values_byte_offset** = 0

- void **set_sparse_values_byte_offset**(value: int)
- int **get_sparse_values_byte_offset**()

The offset relative to the start of the bufferView in bytes.

---

int **type**

- void **set_type**(value: int)
- int **get_type**()

Deprecated: Use accessor_type instead.

The GLTF accessor type as an enum. Use accessor_type instead.

GLTFAnimation

Inherits: Resource **<** RefCounted **<** Object

There is currently no description for this class. Please help us by contributing one!

Tutorials

- Runtime file loading and saving

Properties

bool	loop	`false`
String	original_name	`""`

Methods

Variant	get_additional_data(extension_name: StringName)
void	set_additional_data(extension_name: StringName, additional_data: Variant)

Property Descriptions

bool **loop** = `false` 🔗

- void **set_loop**(value: bool)
- bool **get_loop**()

There is currently no description for this property. Please help us by contributing one!

String **original_name** = `""` 🔗

- void **set_original_name**(value: String)
- String **get_original_name**()

The original name of the animation.

Method Descriptions

Variant **get_additional_data**(extension_name: StringName) 🔗

Gets additional arbitrary data in this **GLTFAnimation** instance. This can be used to keep per-node state data in GLTFDocumentExtension classes, which is important because they are stateless.

The argument should be the GLTFDocumentExtension name (does not have to match the extension name in the GLTF file), and the return value can be anything you set. If nothing was set, the return value is null.

---

void **set_additional_data**(extension_name: StringName, additional_data: Variant) 🔗

Sets additional arbitrary data in this **GLTFAnimation** instance. This can be used to keep per-node state data in GLTFDocumentExtension classes, which is important because they are stateless.

The first argument should be the GLTFDocumentExtension name (does not have to match the extension name in the GLTF file), and the second argument can be anything you want.

GLTFBufferView

Inherits: Resource **<** RefCounted **<** Object

Represents a GLTF buffer view.

Description

GLTFBufferView is a data structure representing GLTF a `bufferView` that would be found in the `"bufferViews"` array. A buffer is a blob of binary data. A buffer view is a slice of a buffer that can be used to identify and extract data from the buffer.

Most custom uses of buffers only need to use the buffer, byte_length, and byte_offset. The byte_stride and indices properties are for more advanced use cases such as interleaved mesh data encoded for the GPU.

Tutorials

- Buffers, BufferViews, and Accessors in Khronos glTF specification [https://github.com/KhronosGroup/glTF-Tutorials/blob/master/gltfTutorial/gltfTutorial_005_BuffersBufferViewsAccessors.md]
- Runtime file loading and saving

Properties

int	buffer	`-1`

int	byte_length	`0`
int	byte_offset	`0`
int	byte_stride	`-1`
bool	indices	`false`
bool	vertex_attributes	`false`

Methods

PackedByteArray	load_buffer_view_data(state: GLTFState) const

---

Property Descriptions

int **buffer** = `-1` 🔗

- void **set_buffer**(value: int)
- int **get_buffer**()

The index of the buffer this buffer view is referencing. If `-1`, this buffer view is not referencing any buffer.

---

int **byte_length** = `0` 🔗

- void **set_byte_length**(value: int)
- int **get_byte_length**()

The length, in bytes, of this buffer view. If `0`, this buffer view is empty.

---

int **byte_offset** = `0` 🔗

- void **set_byte_offset**(value: int)
- int **get_byte_offset**()

The offset, in bytes, from the start of the buffer to the start of this buffer view.

---

int **byte_stride** = `-1` 🔗

- void **set_byte_stride**(value: int)
- int **get_byte_stride**()

The stride, in bytes, between interleaved data. If `-1`, this buffer view is not interleaved.

---

bool **indices** = `false` 🔗

- void **set_indices**(value: bool)
- bool **get_indices**()

True if the GLTFBufferView's OpenGL GPU buffer type is an `ELEMENT_ARRAY_BUFFER` used for vertex indices (integer constant `34963`). False if the buffer type is any other value. See Buffers, BufferViews, and Accessors [https://github.com/KhronosGroup/glTF-Tutorials/blob/master/gltfTutorial/gltfTutorial_005_BuffersBufferViewsAccessors.md] for possible values. This property is set on import and used on export.

---

bool **vertex_attributes** = `false`

- void **set_vertex_attributes**(value: bool)
- bool **get_vertex_attributes**()

True if the GLTFBufferView's OpenGL GPU buffer type is an `ARRAY_BUFFER` used for vertex attributes (integer constant `34962`). False if the buffer type is any other value. See Buffers, BufferViews, and Accessors [https://github.com/KhronosGroup/glTF-Tutorials/blob/master/gltfTutorial/gltfTutorial_005_BuffersBufferViewsAccessors.md] for possible values. This property is set on import and used on export.

Method Descriptions

PackedByteArray **load_buffer_view_data**(state: GLTFState) const

Loads the buffer view data from the buffer referenced by this buffer view in the given GLTFState. Interleaved data with a byte stride is not yet supported by this method. The data is returned as a PackedByteArray.

GLTFCamera

Inherits: Resource **<** RefCounted **<** Object

Represents a GLTF camera.

Description

Represents a camera as defined by the base GLTF spec.

Tutorials

- Runtime file loading and saving
- GLTF camera detailed specification [https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html#reference-camera]
- GLTF camera spec and example file [https://github.com/KhronosGroup/glTF-Tutorials/blob/master/gltfTutorial/gltfTutorial_015_SimpleCameras.md]

Properties

float	depth_far	`4000.0`
float	depth_near	`0.05`
float	fov	`1.309`

bool	perspective	`true`
float	size_mag	`0.5`

Methods

GLTFCamera	from_dictionary(dictionary: Dictionary) static
GLTFCamera	from_node(camera_node: Camera3D) static
Dictionary	to_dictionary() const
Camera3D	to_node() const

Property Descriptions

float **depth_far** = `4000.0` 🔗

- void **set_depth_far**(value: float)
- float **get_depth_far**()

The distance to the far culling boundary for this camera relative to its local Z axis, in meters. This maps to GLTF's `zfar` property.

---

float **depth_near** = `0.05` 🔗

- void **set_depth_near**(value: float)
- float **get_depth_near**()

The distance to the near culling boundary for this camera relative to its local Z axis, in meters. This maps to GLTF's `znear` property.

---

float **fov** = `1.309`

- void **set_fov**(value: float)
- float **get_fov**()

The FOV of the camera. This class and GLTF define the camera FOV in radians, while Godot uses degrees. This maps to GLTF's `yfov` property. This value is only used for perspective cameras, when perspective is true.

---

bool **perspective** = `true`

- void **set_perspective**(value: bool)
- bool **get_perspective**()

Whether or not the camera is in perspective mode. If false, the camera is in orthographic/orthogonal mode. This maps to GLTF's camera `type` property. See Camera3D.projection and the GLTF spec for more information.

---

float **size_mag** = `0.5`

- void **set_size_mag**(value: float)
- float **get_size_mag**()

The size of the camera. This class and GLTF define the camera size magnitude as a radius in meters, while Godot defines it as a diameter in meters. This maps to GLTF's `ymag` property. This value is only used for orthographic/orthogonal cameras, when perspective is false.

Method Descriptions

GLTFCamera **from_dictionary**(dictionary: Dictionary) static 🔗

Creates a new GLTFCamera instance by parsing the given Dictionary.

GLTFCamera **from_node**(camera_node: Camera3D) static 🔗

Create a new GLTFCamera instance from the given Godot Camera3D node.

Dictionary **to_dictionary**() const 🔗

Serializes this GLTFCamera instance into a Dictionary.

Camera3D **to_node**() const 🔗

Converts this GLTFCamera instance into a Godot Camera3D node.

GLTFDocument

Inherits: Resource **<** RefCounted **<** Object

Inherited By: FBXDocument

Class for importing and exporting glTF files in and out of Godot.

Description

GLTFDocument supports reading data from a glTF file, buffer, or Godot scene. This data can then be written to the filesystem, buffer, or used to create a Godot scene.

All of the data in a GLTF scene is stored in the GLTFState class. GLTFDocument processes state objects, but does not contain any scene data itself. GLTFDocument has member variables to store export configuration settings such as the image format, but is otherwise stateless. Multiple scenes can be processed with the same settings using the same GLTFDocument object and different GLTFState objects.

GLTFDocument can be extended with arbitrary functionality by extending the GLTFDocumentExtension class and registering it with GLTFDocument via register_gltf_document_extension. This allows for custom data to be imported and exported.

Tutorials

- Runtime file loading and saving
- glTF 'What the duck?' guide [https://www.khronos.org/files/gltf20-reference-guide.pdf]

- Khronos glTF specification [https://registry.khronos.org/glTF/]

Properties

String	image_format	`"PNG"`
float	lossy_quality	`0.75`
RootNodeMode	root_node_mode	`0`

Methods

Error	append_from_buffer(bytes: PackedByteArray, base_path: String, state: GLTFState, flags: int = 0)
Error	append_from_file(path: String, state: GLTFState, flags: int = 0, base_path: String = "")
Error	append_from_scene(node: Node, state: GLTFState, flags: int = 0)
PackedByteArray	generate_buffer(state: GLTFState)
Node	generate_scene(state: GLTFState, bake_fps: float = 30, trimming: bool = false,

	remove_immutable_tracks: bool = true)
void	register_gltf_document_extension(extension: GLTFDocumentExtension, first_priority: bool = false) static
void	unregister_gltf_document_extension(extension: GLTFDocumentExtension) static
Error	write_to_filesystem(state: GLTFState, path: String)

Enumerations

enum **RootNodeMode**: 🔗

RootNodeMode **ROOT_NODE_MODE_SINGLE_ROOT** = `0`

Treat the Godot scene's root node as the root node of the glTF file, and mark it as the single root node via the `GODOT_single_root` glTF extension. This will be parsed the same as ROOT_NODE_MODE_KEEP_ROOT if the implementation does not support `GODOT_single_root`.

RootNodeMode **ROOT_NODE_MODE_KEEP_ROOT** = `1`

Treat the Godot scene's root node as the root node of the glTF file, but do not mark it as anything special. An extra root node will be generated when importing into Godot. This uses only vanilla glTF features. This is equivalent to the behavior in Godot 4.1 and earlier.

RootNodeMode **ROOT_NODE_MODE_MULTI_ROOT** = `2`

Treat the Godot scene's root node as the name of the glTF scene, and add all of its children as root nodes of the glTF file. This uses only vanilla glTF features. This avoids an extra root node, but only the name of the Godot scene's root node will be preserved, as it will not be saved as a node.

Property Descriptions

String **image_format** = `"PNG"`

- void **set_image_format**(value: String)
- String **get_image_format**()

The user-friendly name of the export image format. This is used when exporting the GLTF file, including writing to a file and writing to a byte array.

By default, Godot allows the following options: "None", "PNG", "JPEG", "Lossless WebP", and "Lossy WebP". Support for more image formats can be added in GLTFDocumentExtension classes.

---

float **lossy_quality** = `0.75`

- void **set_lossy_quality**(value: float)
- float **get_lossy_quality**()

If image_format is a lossy image format, this determines the lossy quality of the image. On a range of `0.0` to `1.0`, where `0.0` is the lowest quality and `1.0` is the highest quality. A lossy quality of `1.0` is not the same as lossless.

RootNodeMode **root_node_mode** = `0`

- void **set_root_node_mode**(value: RootNodeMode)
- RootNodeMode **get_root_node_mode**()

How to process the root node during export. See RootNodeMode for details. The default and recommended value is ROOT_NODE_MODE_SINGLE_ROOT.

Note: Regardless of how the glTF file is exported, when importing, the root node type and name can be overridden in the scene import settings tab.

Method Descriptions

Error **append_from_buffer**(bytes: PackedByteArray, base_path: String, state: GLTFState, flags: int = 0)

Takes a PackedByteArray defining a GLTF and imports the data to the given GLTFState object through the `state` parameter.

Note: The `base_path` tells append_from_buffer where to find dependencies and can be empty.

Error **append_from_file**(path: String, state: GLTFState, flags: int = 0, base_path: String = "")

Takes a path to a GLTF file and imports the data at that file path to the given GLTFState object through the `state` parameter.

Note: The `base_path` tells append_from_file where to find dependencies and can be empty.

Error **append_from_scene**(node: Node, state: GLTFState, flags: int = 0) 🔗

Takes a Godot Engine scene node and exports it and its descendants to the given GLTFState object through the `state` parameter.

PackedByteArray **generate_buffer**(state: GLTFState) 🔗

Takes a GLTFState object through the `state` parameter and returns a GLTF PackedByteArray.

Node **generate_scene**(state: GLTFState, bake_fps: float = 30, trimming: bool = false, remove_immutable_tracks: bool = true) 🔗

Takes a GLTFState object through the `state` parameter and returns a Godot Engine scene node.

The `bake_fps` parameter overrides the bake_fps in `state`.

---

void **register_gltf_document_extension**(extension: GLTFDocumentExtension, first_priority: bool = false) static 🔗

Registers the given GLTFDocumentExtension instance with GLTFDocument. If `first_priority` is true, this extension will be run first. Otherwise, it will be run last.

Note: Like GLTFDocument itself, all GLTFDocumentExtension classes must be stateless in order to function properly. If you need to store data, use the `set_additional_data` and `get_additional_data` methods in GLTFState or GLTFNode.

---

void **unregister_gltf_document_extension**(extension: GLTFDocumentExtension) static

Unregisters the given GLTFDocumentExtension instance.

Error **write_to_filesystem**(state: GLTFState, path: String)

Takes a GLTFState object through the `state` parameter and writes a glTF file to the filesystem.

Note: The extension of the glTF file determines if it is a .glb binary file or a .gltf text file.

GLTFDocumentExtension

Inherits: Resource **<** RefCounted **<** Object

Inherited By: GLTFDocumentExtensionConvertImporterMesh

GLTFDocument extension class.

Description

Extends the functionality of the GLTFDocument class by allowing you to run arbitrary code at various stages of GLTF import or export.

To use, make a new class extending GLTFDocumentExtension, override any methods you need, make an instance of your class, and register it using GLTFDocument.register_gltf_document_extension.

Note: Like GLTFDocument itself, all GLTFDocumentExtension classes must be stateless in order to function properly. If you need to store data, use the `set_additional_data` and `get_additional_data` methods in GLTFState or GLTFNode.

Tutorials

- Runtime file loading and saving

Methods

void	_convert_scene_node(state: GLTFState, gltf_node: GLTFNode, scene_node: Node)

	virtual
Error	_export_node(state: GLTFState, gltf_node: GLTFNode, json: Dictionary, node: Node) virtual
Error	_export_post(state: GLTFState) virtual
Error	_export_preflight(state: GLTFState, root: Node) virtual
Error	_export_preserialize(state: GLTFState) virtual
Node3D	_generate_scene_node(state: GLTFState, gltf_node: GLTFNode, scene_parent: Node) virtual
String	_get_image_file_extension() virtual
PackedStringArray	_get_saveable_image_formats() virtual
PackedStringArray	_get_supported_extensions() virtual
Error	_import_node(state: GLTFState, gltf_node: GLTFNode, json: Dictionary, node: Node) virtual
Error	_import_post(state: GLTFState, root: Node) virtual

Error	_import_post_parse(state: GLTFState) virtual
Error	_import_preflight(state: GLTFState, extensions: PackedStringArray) virtual
Error	_parse_image_data(state: GLTFState, image_data: PackedByteArray, mime_type: String, ret_image: Image) virtual
Error	_parse_node_extensions(state: GLTFState, gltf_node: GLTFNode, extensions: Dictionary) virtual
Error	_parse_texture_json(state: GLTFState, texture_json: Dictionary, ret_gltf_texture: GLTFTexture) virtual
Error	_save_image_at_path(state: GLTFState, image: Image, file_path: String, image_format: String, lossy_quality: float) virtual
PackedByteArray	_serialize_image_to_bytes(state: GLTFState, image: Image, image_dict: Dictionary, image_format: String, lossy_quality: float) virtual
Error	_serialize_texture_json(state: GLTFState, texture_json: Dictionary, gltf_texture: GLTFTexture, image_format: String) virtual

---

Method Descriptions

void **_convert_scene_node**(state: GLTFState, gltf_node: GLTFNode, scene_node: Node) virtual

Part of the export process. This method is run after _export_preflight and before _export_preserialize.

Runs when converting the data from a Godot scene node. This method can be used to process the Godot scene node data into a format that can be used by _export_node.

Error **_export_node**(state: GLTFState, gltf_node: GLTFNode, json: Dictionary, node: Node) virtual

Part of the export process. This method is run after _get_saveable_image_formats and before _export_post. If this **GLTFDocumentExtension** is used for exporting images, this runs after _serialize_texture_json.

This method can be used to modify the final JSON of each node. Data should be primarily stored in `gltf_node` prior to serializing the JSON, but the original Godot `node` is also provided if available. The node may be null if not available, such as when exporting GLTF data not generated from a Godot scene.

Error **_export_post**(state: GLTFState) virtual

Part of the export process. This method is run last, after all other parts of the export process.

This method can be used to modify the final JSON of the generated GLTF file.

Error **_export_preflight**(state: GLTFState, root: Node) virtual

Part of the export process. This method is run first, before all other parts of the export process.

The return value is used to determine if this **GLTFDocumentExtension** instance should be used for exporting a given GLTF file. If @GlobalScope.OK, the export will use this **GLTFDocumentExtension** instance. If not overridden, @GlobalScope.OK is returned.

Error **_export_preserialize**(state: GLTFState) virtual

Part of the export process. This method is run after _convert_scene_node and before _get_saveable_image_formats.

This method can be used to alter the state before performing serialization. It runs every time when generating a buffer with GLTFDocument.generate_buffer or writing to the file system with GLTFDocument.write_to_filesystem.

Node3D **_generate_scene_node**(state: GLTFState, gltf_node: GLTFNode, scene_parent: Node) virtual

Part of the import process. This method is run after _import_post_parse and before _import_node.

Runs when generating a Godot scene node from a GLTFNode. The returned node will be added to the scene tree. Multiple nodes can be generated in this step if they are added as a child of the returned node.

Note: The `scene_parent` parameter may be null if this is the single root node.

String **_get_image_file_extension**() virtual

Returns the file extension to use for saving image data into, for example, `".png"`. If defined, when this extension is used to handle images, and the images are saved to a separate file, the image bytes will be copied to a file with this extension. If this is set, there should be a ResourceImporter class able to import the file. If not defined or empty, Godot will save the image into a PNG file.

PackedStringArray **_get_saveable_image_formats**() virtual

Part of the export process. This method is run after _convert_scene_node and before _export_node.

Returns an array of the image formats that can be saved/exported by this extension. This extension will only be selected as the image exporter if the GLTFDocument's GLTFDocument.image_format is in this array. If this **GLTFDocumentExtension** is selected as the image exporter, one of the _save_image_at_path or _serialize_image_to_bytes methods will run next, otherwise _export_node will run next. If the format name contains `"Lossy"`, the lossy quality slider will be displayed.

PackedStringArray **_get_supported_extensions**() virtual

Part of the import process. This method is run after _import_preflight and before _parse_node_extensions.

Returns an array of the GLTF extensions supported by this GLTFDocumentExtension class. This is used to validate if a GLTF file with required extensions can be loaded.

Error **_import_node**(state: GLTFState, gltf_node: GLTFNode, json: Dictionary, node: Node) virtual

Part of the import process. This method is run after _generate_scene_node and before _import_post.

This method can be used to make modifications to each of the generated Godot scene nodes.

Error **_import_post**(state: GLTFState, root: Node) virtual

Part of the import process. This method is run last, after all other parts of the import process.

This method can be used to modify the final Godot scene generated by the import process.

Error **_import_post_parse**(state: GLTFState) virtual

Part of the import process. This method is run after _parse_node_extensions and before _generate_scene_node.

This method can be used to modify any of the data imported so far after parsing, before generating the nodes and then running the final per-node import step.

Error **_import_preflight**(state: GLTFState, extensions: PackedStringArray) virtual

Part of the import process. This method is run first, before all other parts of the import process.

The return value is used to determine if this **GLTFDocumentExtension** instance should be used for importing a given GLTF file. If @GlobalScope.OK, the import will use this **GLTFDocumentExtension** instance. If not overridden, @GlobalScope.OK is returned.

Error **_parse_image_data**(state: GLTFState, image_data: PackedByteArray, mime_type: String, ret_image: Image) virtual 🔗

Part of the import process. This method is run after _parse_node_extensions and before _parse_texture_json.

Runs when parsing image data from a GLTF file. The data could be sourced from a separate file, a URI, or a buffer, and then is passed as a byte array.

Error **_parse_node_extensions**(state: GLTFState, gltf_node: GLTFNode, extensions: Dictionary) virtual 🔗

Part of the import process. This method is run after _get_supported_extensions and before _import_post_parse.

Runs when parsing the node extensions of a GLTFNode. This method can be used to process the extension JSON data into a format that can be used by _generate_scene_node. The return value should be a member of the Error enum.

Error **_parse_texture_json**(state: GLTFState, texture_json: Dictionary, ret_gltf_texture: GLTFTexture) virtual 🔗

Part of the import process. This method is run after _parse_image_data and before _generate_scene_node.

Runs when parsing the texture JSON from the GLTF textures array. This can be used to set the source image index to use as the texture.

Error **_save_image_at_path**(state: GLTFState, image: Image, file_path: String, image_format: String, lossy_quality: float) virtual 🔗

Part of the export process. This method is run after _get_saveable_image_formats and before _serialize_texture_json.

This method is run when saving images separately from the GLTF file. When images are embedded, _serialize_image_to_bytes runs instead. Note that these methods only run when this **GLTFDocumentExtension** is selected as the image exporter.

PackedByteArray **_serialize_image_to_bytes**(state: GLTFState, image: Image, image_dict: Dictionary, image_format: String, lossy_quality: float) virtual 🔗

Part of the export process. This method is run after _get_saveable_image_formats and before _serialize_texture_json.

This method is run when embedding images in the GLTF file. When images are saved separately, _save_image_at_path runs instead. Note that these methods only run when this **GLTFDocumentExtension** is selected as the image exporter.

This method must set the image MIME type in the `image_dict` with the `"mimeType"` key. For example, for a PNG image, it would be set to `"image/png"`. The return value must be a PackedByteArray containing the image data.

Error **_serialize_texture_json**(state: GLTFState, texture_json: Dictionary, gltf_texture: GLTFTexture, image_format: String) virtual 🔗

Part of the export process. This method is run after _save_image_at_path or _serialize_image_to_bytes, and before _export_node. Note that this method only runs when this **GLTFDocumentExtension** is selected as the image exporter.

This method can be used to set up the extensions for the texture JSON by editing `texture_json`. The extension must also be added as used extension with GLTFState.add_used_extension, be sure to set `required` to `true` if you are not providing a fallback.

GLTFDocumentExtensionConvertImporterMesh

Inherits: GLTFDocumentExtension **<** Resource **<** RefCounted **<** Object

There is currently no description for this class. Please help us by contributing one!

Tutorials

- Runtime file loading and saving

GLTFLight

Inherits: Resource **<** RefCounted **<** Object

Represents a GLTF light.

Description

Represents a light as defined by the `KHR_lights_punctual` GLTF extension.

Tutorials

- Runtime file loading and saving
- KHR_lights_punctual GLTF extension spec [https://github.com/KhronosGroup/glTF/blob/main/extensions/2.0/Khronos/KHR_lights_punctual]

Properties

Color	color	`Color(1, 1, 1, 1)`
float	inner_cone_angle	`0.0`
float	intensity	`1.0`
String	light_type	`""`

float	outer_cone_angle	`0.785398`
float	range	`inf`

Methods

GLTFLight	from_dictionary(dictionary: Dictionary) static
GLTFLight	from_node(light_node: Light3D) static
Variant	get_additional_data(extension_name: StringName)
void	set_additional_data(extension_name: StringName, additional_data: Variant)
Dictionary	to_dictionary() const
Light3D	to_node() const

Property Descriptions

Color **color** = `Color(1, 1, 1, 1)` 🔗

- void **set_color**(value: Color)

- Color **get_color**()

The Color of the light. Defaults to white. A black color causes the light to have no effect.

---

float **inner_cone_angle** = `0.0`

- void **set_inner_cone_angle**(value: float)
- float **get_inner_cone_angle**()

The inner angle of the cone in a spotlight. Must be less than or equal to the outer cone angle.

Within this angle, the light is at full brightness. Between the inner and outer cone angles, there is a transition from full brightness to zero brightness. When creating a Godot SpotLight3D, the ratio between the inner and outer cone angles is used to calculate the attenuation of the light.

---

float **intensity** = `1.0`

- void **set_intensity**(value: float)
- float **get_intensity**()

The intensity of the light. This is expressed in candelas (lumens per steradian) for point and spot lights, and lux (lumens per m²) for directional lights. When creating a Godot light, this value is converted to a unitless multiplier.

String **light_type** = `""`

- void **set_light_type**(value: String)
- String **get_light_type**()

The type of the light. The values accepted by Godot are "point", "spot", and "directional", which correspond to Godot's OmniLight3D, SpotLight3D, and DirectionalLight3D respectively.

---

float **outer_cone_angle** = `0.785398` 🔗

- void **set_outer_cone_angle**(value: float)
- float **get_outer_cone_angle**()

The outer angle of the cone in a spotlight. Must be greater than or equal to the inner angle.

At this angle, the light drops off to zero brightness. Between the inner and outer cone angles, there is a transition from full brightness to zero brightness. If this angle is a half turn, then the spotlight emits in all directions. When creating a Godot SpotLight3D, the outer cone angle is used as the angle of the spotlight.

---

float **range** = `inf` 🔗

- void **set_range**(value: float)
- float **get_range**()

The range of the light, beyond which the light has no effect. GLTF lights with no range defined behave like physical lights (which have infinite range). When creating a Godot light, the range is clamped to 4096.

Method Descriptions

GLTFLight **from_dictionary**(dictionary: Dictionary) static 🔗

Creates a new GLTFLight instance by parsing the given Dictionary.

GLTFLight **from_node**(light_node: Light3D) static

Create a new GLTFLight instance from the given Godot Light3D node.

Variant **get_additional_data**(extension_name: StringName)

There is currently no description for this method. Please help us by contributing one!

---

void **set_additional_data**(extension_name: StringName, additional_data: Variant)

There is currently no description for this method. Please help us by contributing one!

Dictionary **to_dictionary**() const

Serializes this GLTFLight instance into a Dictionary.

Light3D **to_node**() const

Converts this GLTFLight instance into a Godot Light3D node.

GLTFMesh

Inherits: Resource **<** RefCounted **<** Object

GLTFMesh represents a GLTF mesh.

Description

GLTFMesh handles 3D mesh data imported from GLTF files. It includes properties for blend channels, blend weights, instance materials, and the mesh itself.

Tutorials

- Runtime file loading and saving

Properties

PackedFloat32Array	blend_weights	`PackedFloat32Array()`
Array[Material]	instance_materials	`[]`
ImporterMesh	mesh	
String	original_name	`""`

Methods

Variant	get_additional_data(extension_name: StringName)
void	set_additional_data(extension_name: StringName, additional_data: Variant)

Property Descriptions

PackedFloat32Array **blend_weights** = `PackedFloat32Array()`

- void **set_blend_weights**(value: PackedFloat32Array)
- PackedFloat32Array **get_blend_weights**()

An array of floats representing the blend weights of the mesh.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedFloat32Array for more details.

Array[Material] **instance_materials** = `[]`

- void **set_instance_materials**(value: Array[Material])
- Array[Material] **get_instance_materials**()

An array of Material objects representing the materials used in the mesh.

ImporterMesh **mesh**

- void **set_mesh**(value: ImporterMesh)
- ImporterMesh **get_mesh**()

The ImporterMesh object representing the mesh itself.

String **original_name** = `""` 🔗

- void **set_original_name**(value: String)
- String **get_original_name**()

The original name of the mesh.

Method Descriptions

Variant **get_additional_data**(extension_name: StringName) 🔗

Gets additional arbitrary data in this **GLTFMesh** instance. This can be used to keep per-node state data in GLTFDocumentExtension classes, which is important because they are stateless.

The argument should be the GLTFDocumentExtension name (does not have to match the extension name in the GLTF file), and the return value can be anything you set. If nothing was set, the return value is null.

---

void **set_additional_data**(extension_name: StringName, additional_data: Variant) 🔗

Sets additional arbitrary data in this **GLTFMesh** instance. This can be used to keep per-node state data in GLTFDocumentExtension classes, which is important because they are stateless.

The first argument should be the GLTFDocumentExtension name (does not have to match the extension name in the GLTF file), and the second argument can be anything you want.

GLTFNode

Inherits: Resource **<** RefCounted **<** Object

GLTF node class.

Description

Represents a GLTF node. GLTF nodes may have names, transforms, children (other GLTF nodes), and more specialized properties (represented by their own classes).

GLTF nodes generally exist inside of GLTFState which represents all data of a GLTF file. Most of GLTFNode's properties are indices of other data in the GLTF file. You can extend a GLTF node with additional properties by using get_additional_data and set_additional_data.

Tutorials

- Runtime file loading and saving
- GLTF scene and node spec [https://github.com/KhronosGroup/glTF-Tutorials/blob/master/gltfTutorial/gltfTutorial_004_ScenesNodes.md"]

Properties

int	camera	`-1`
PackedInt32Array	children	`PackedInt32Array()`

int	height	`-1`
int	light	`-1`
int	mesh	`-1`
String	original_name	`""`
int	parent	`-1`
Vector3	position	`Vector3(0, 0, 0)`
Quaternion	rotation	`Quaternion(0, 0, 0, 1)`
Vector3	scale	`Vector3(1, 1, 1)`
int	skeleton	`-1`
int	skin	`-1`
Transform3D	xform	`Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)`

Methods

Variant	get_additional_data(extension_name: StringName)
void	set_additional_data(extension_name: StringName, additional_data: Variant)

Property Descriptions

int **camera** = `-1` 🔗

- void **set_camera**(value: int)
- int **get_camera**()

If this GLTF node is a camera, the index of the GLTFCamera in the GLTFState that describes the camera's properties. If -1, this node is not a camera.

PackedInt32Array **children** = `PackedInt32Array()` 🔗

- void **set_children**(value: PackedInt32Array)
- PackedInt32Array **get_children**()

The indices of the child nodes in the GLTFState. If this GLTF node has no children, this will be an empty array.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedInt32Array for more details.

---

int **height** = `-1` 🔗

- void **set_height**(value: int)
- int **get_height**()

How deep into the node hierarchy this node is. A root node will have a height of 0, its children will have a height of 1, and so on. If -1, the height has not been calculated.

int **light** = `-1`

- void **set_light**(value: int)
- int **get_light**()

If this GLTF node is a light, the index of the GLTFLight in the GLTFState that describes the light's properties. If -1, this node is not a light.

int **mesh** = `-1`

- void **set_mesh**(value: int)
- int **get_mesh**()

If this GLTF node is a mesh, the index of the GLTFMesh in the GLTFState that describes the mesh's properties. If -1, this node is not a mesh.

String **original_name** = `""`

- void **set_original_name**(value: String)
- String **get_original_name**()

The original name of the node.

int **parent** = `-1`

- void **set_parent**(value: int)
- int **get_parent**()

The index of the parent node in the GLTFState. If -1, this node is a root node.

Vector3 **position** = `Vector3(0, 0, 0)`

- void **set_position**(value: Vector3)
- Vector3 **get_position**()

The position of the GLTF node relative to its parent.

Quaternion **rotation** = `Quaternion(0, 0, 0, 1)`

- void **set_rotation**(value: Quaternion)
- Quaternion **get_rotation**()

The rotation of the GLTF node relative to its parent.

Vector3 **scale** = `Vector3(1, 1, 1)`

- void **set_scale**(value: Vector3)
- Vector3 **get_scale**()

The scale of the GLTF node relative to its parent.

---

int **skeleton** = `-1`

- void **set_skeleton**(value: int)
- int **get_skeleton**()

If this GLTF node has a skeleton, the index of the GLTFSkeleton in the GLTFState that describes the skeleton's properties. If -1, this node does not have a skeleton.

---

int **skin** = `-1` 🔗

- void **set_skin**(value: int)
- int **get_skin**()

If this GLTF node has a skin, the index of the GLTFSkin in the GLTFState that describes the skin's properties. If -1, this node does not have a skin.

Transform3D **xform** = `Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)` 🔗

- void **set_xform**(value: Transform3D)
- Transform3D **get_xform**()

The transform of the GLTF node relative to its parent. This property is usually unused since the position, rotation, and scale properties are preferred.

Method Descriptions

Variant **get_additional_data**(extension_name: StringName) 🔗

Gets additional arbitrary data in this **GLTFNode** instance. This can be used to keep per-node state data in GLTFDocumentExtension classes, which is important because they are stateless.

The argument should be the GLTFDocumentExtension name (does not have to match the extension name in the GLTF file), and the return value can be anything you set. If nothing was set, the return value is null.

void **set_additional_data**(extension_name: StringName, additional_data: Variant) 🔗

Sets additional arbitrary data in this **GLTFNode** instance. This can be used to keep per-node state data in GLTFDocumentExtension classes, which is important because they are stateless.

The first argument should be the GLTFDocumentExtension name (does not have to match the extension name in the GLTF file), and the second argument can be anything you want.

GLTFPhysicsBody

Inherits: Resource **<** RefCounted **<** Object

Represents a GLTF physics body.

Description

Represents a physics body as an intermediary between the `OMI_physics_body` GLTF data and Godot's nodes, and it's abstracted in a way that allows adding support for different GLTF physics extensions in the future.

Tutorials

- Runtime file loading and saving
- OMI_physics_body GLTF extension [https://github.com/omigroup/gltf-extensions/tree/main/extensions/2.0/OMI_physics_body]

Properties

Vector3	angular_velocity	`Vector3(0, 0, 0)`
String	body_type	`"rigid"`
Vector3	center_of_mass	`Vector3(0, 0, 0)`

Vector3	inertia_diagonal	`Vector3(0, 0, 0)`
Quaternion	inertia_orientation	`Quaternion(0, 0, 0, 1)`
Basis	inertia_tensor	`Basis(0, 0, 0, 0, 0, 0, 0, 0, 0)`
Vector3	linear_velocity	`Vector3(0, 0, 0)`
float	mass	`1.0`

Methods

GLTFPhysicsBody	from_dictionary(dictionary: Dictionary) static
GLTFPhysicsBody	from_node(body_node: CollisionObject3D) static
Dictionary	to_dictionary() const
CollisionObject3D	to_node() const

Property Descriptions

Vector3 **angular_velocity** = `Vector3(0, 0, 0)` 🔗

- void **set_angular_velocity**(value: Vector3)
- Vector3 **get_angular_velocity**()

The angular velocity of the physics body, in radians per second. This is only used when the body type is "rigid" or "vehicle".

String **body_type** = `"rigid"`

- void **set_body_type**(value: String)
- String **get_body_type**()

The type of the body. When importing, this controls what type of CollisionObject3D node Godot should generate. Valid values are "static", "animatable", "character", "rigid", "vehicle", and "trigger". When exporting, this will be squashed down to one of "static", "kinematic", or "dynamic" motion types, or the "trigger" property.

Vector3 **center_of_mass** = `Vector3(0, 0, 0)`

- void **set_center_of_mass**(value: Vector3)
- Vector3 **get_center_of_mass**()

The center of mass of the body, in meters. This is in local space relative to the body. By default, the center of the mass is the body's origin.

Vector3 **inertia_diagonal** = `Vector3(0, 0, 0)`

- void **set_inertia_diagonal**(value: Vector3)
- Vector3 **get_inertia_diagonal**()

The inertia strength of the physics body, in kilogram meter squared ($kg \cdot m^2$). This represents the inertia around the principle axes, the diagonal of the inertia tensor matrix. This is only used when the body type is "rigid" or "vehicle".

When converted to a Godot RigidBody3D node, if this value is zero, then the inertia will be calculated automatically.

Quaternion **inertia_orientation** = `Quaternion(0, 0, 0, 1)` 🔗

- void **set_inertia_orientation**(value: Quaternion)
- Quaternion **get_inertia_orientation**()

The inertia orientation of the physics body. This defines the rotation of the inertia's principle axes relative to the object's local axes. This is only used when the body type is "rigid" or "vehicle" and inertia_diagonal is set to a non-zero value.

Basis **inertia_tensor** = `Basis(0, 0, 0, 0, 0, 0, 0, 0, 0)` 🔗

- void **set_inertia_tensor**(value: Basis)
- Basis **get_inertia_tensor**()

Deprecated: This property may be changed or removed in future versions.

The inertia tensor of the physics body, in kilogram meter squared (kg·m²). This is only used when the body type is "rigid" or "vehicle".

When converted to a Godot RigidBody3D node, if this value is zero, then the inertia will be calculated automatically.

Vector3 **linear_velocity** = `Vector3(0, 0, 0)` 🔗

- void **set_linear_velocity**(value: Vector3)
- Vector3 **get_linear_velocity**()

The linear velocity of the physics body, in meters per second. This is only used when the body type is "rigid" or "vehicle".

---

float **mass** = `1.0`

- void **set_mass**(value: float)
- float **get_mass**()

The mass of the physics body, in kilograms. This is only used when the body type is "rigid" or "vehicle".

Method Descriptions

GLTFPhysicsBody **from_dictionary**(dictionary: Dictionary) static

Creates a new GLTFPhysicsBody instance by parsing the given Dictionary in the `OMI_physics_body` GLTF extension format.

GLTFPhysicsBody **from_node**(body_node: CollisionObject3D) static

Creates a new GLTFPhysicsBody instance from the given Godot CollisionObject3D node.

Dictionary **to_dictionary**() const

Serializes this GLTFPhysicsBody instance into a Dictionary. It will be in the format expected by the `OMI_physics_body` GLTF extension.

CollisionObject3D **to_node**() const

Converts this GLTFPhysicsBody instance into a Godot CollisionObject3D node.

GLTFPhysicsShape

Inherits: Resource **<** RefCounted **<** Object

Represents a GLTF physics shape.

Description

Represents a physics shape as defined by the `OMI_physics_shape` or `OMI_collider` GLTF extensions. This class is an intermediary between the GLTF data and Godot's nodes, and it's abstracted in a way that allows adding support for different GLTF physics extensions in the future.

Tutorials

- Runtime file loading and saving
- OMI_physics_shape GLTF extension [https://github.com/omigroup/gltf-extensions/tree/main/extensions/2.0/OMI_physics_shape]
- OMI_collider GLTF extension [https://github.com/omigroup/gltf-extensions/tree/main/extensions/2.0/Archived/OMI_collider]

Properties

float	height	`2.0`
ImporterMesh	importer_mesh	

bool	is_trigger	`false`
int	mesh_index	`-1`
float	radius	`0.5`
String	shape_type	`""`
Vector3	size	`Vector3(1, 1, 1)`

Methods

GLTFPhysicsShape	from_dictionary(dictionary: Dictionary) static
GLTFPhysicsShape	from_node(shape_node: CollisionShape3D) static
GLTFPhysicsShape	from_resource(shape_resource: Shape3D) static
Dictionary	to_dictionary() const
CollisionShape3D	to_node(cache_shapes: bool = false)
Shape3D	to_resource(cache_shapes: bool = false)

Property Descriptions

float **height** = `2.0` 🔗

- void **set_height**(value: float)
- float **get_height**()

The height of the shape, in meters. This is only used when the shape type is "capsule" or "cylinder". This value should not be negative, and for "capsule" it should be at least twice the radius.

ImporterMesh **importer_mesh** 🔗

- void **set_importer_mesh**(value: ImporterMesh)
- ImporterMesh **get_importer_mesh**()

The ImporterMesh resource of the shape. This is only used when the shape type is "hull" (convex hull) or "trimesh" (concave trimesh).

---

bool **is_trigger** = `false` 🔗

- void **set_is_trigger**(value: bool)
- bool **get_is_trigger**()

If `true`, indicates that this shape is a trigger. For Godot, this means that the shape should be a child of an Area3D node.

This is the only variable not used in the to_node method, it's intended to be used alongside when deciding where to add the generated node as a child.

---

int **mesh_index** = `-1`

- void **set_mesh_index**(value: int)
- int **get_mesh_index**()

The index of the shape's mesh in the GLTF file. This is only used when the shape type is "hull" (convex hull) or "trimesh" (concave trimesh).

---

float **radius** = `0.5`

- void **set_radius**(value: float)
- float **get_radius**()

The radius of the shape, in meters. This is only used when the shape type is "capsule", "cylinder", or "sphere". This value should not be negative.

String **shape_type** = `""`

- void **set_shape_type**(value: String)
- String **get_shape_type**()

The type of shape this shape represents. Valid values are "box", "capsule", "cylinder", "sphere", "hull", and "trimesh".

Vector3 **size** = `Vector3(1, 1, 1)`

- void **set_size**(value: Vector3)
- Vector3 **get_size**()

The size of the shape, in meters. This is only used when the shape type is "box", and it represents the "diameter" of the box. This value should not be negative.

Method Descriptions

GLTFPhysicsShape **from_dictionary**(dictionary: Dictionary) static

Creates a new GLTFPhysicsShape instance by parsing the given Dictionary.

GLTFPhysicsShape **from_node**(shape_node: CollisionShape3D) static

Creates a new GLTFPhysicsShape instance from the given Godot CollisionShape3D node.

GLTFPhysicsShape **from_resource**(shape_resource: Shape3D) static

Creates a new GLTFPhysicsShape instance from the given Godot Shape3D resource.

Dictionary **to_dictionary**() const

Serializes this GLTFPhysicsShape instance into a Dictionary in the format defined by `OMI_physics_shape`.

CollisionShape3D **to_node**(cache_shapes: bool = false)

Converts this GLTFPhysicsShape instance into a Godot CollisionShape3D node.

Shape3D **to_resource**(cache_shapes: bool = false) 🔗

Converts this GLTFPhysicsShape instance into a Godot Shape3D resource.

GLTFSkeleton

Inherits: Resource **<** RefCounted **<** Object

There is currently no description for this class. Please help us by contributing one!

Tutorials

- Runtime file loading and saving

Properties

PackedInt32Array	joints	`PackedInt32Array()`
PackedInt32Array	roots	`PackedInt32Array()`

Methods

BoneAttachment3D	get_bone_attachment(idx: int)
int	get_bone_attachment_count()
Dictionary	get_godot_bone_node()

Skeleton3D	get_godot_skeleton()
Array[String]	get_unique_names()
void	set_godot_bone_node(godot_bone_node: Dictionary)
void	set_unique_names(unique_names: Array[String])

Property Descriptions

PackedInt32Array **joints** = `PackedInt32Array()` 🔗

- void **set_joints**(value: PackedInt32Array)
- PackedInt32Array **get_joints**()

There is currently no description for this property. Please help us by contributing one!

PackedInt32Array **roots** = `PackedInt32Array()` 🔗

- void **set_roots**(value: PackedInt32Array)
- PackedInt32Array **get_roots**()

There is currently no description for this property. Please help us by contributing one!

Method Descriptions

BoneAttachment3D **get_bone_attachment**(idx: int)

There is currently no description for this method. Please help us by contributing one!

---

int **get_bone_attachment_count**()

There is currently no description for this method. Please help us by contributing one!

Dictionary **get_godot_bone_node**()

Returns a Dictionary that maps skeleton bone indices to the indices of GLTF nodes. This property is unused during import, and only set during export. In a GLTF file, a bone is a node, so Godot converts skeleton bones to GLTF nodes.

Skeleton3D **get_godot_skeleton**()

There is currently no description for this method. Please help us by contributing one!

Array[String] **get_unique_names**()

There is currently no description for this method. Please help us by contributing one!

---

void **set_godot_bone_node**(godot_bone_node: Dictionary)

Sets a Dictionary that maps skeleton bone indices to the indices of GLTF nodes. This property is unused during import, and only set during export. In a GLTF file, a bone is a node, so Godot converts skeleton bones to GLTF nodes.

---

void **set_unique_names**(unique_names: Array[String]) 🔗

There is currently no description for this method. Please help us by contributing one!

GLTFSkin

Inherits: Resource **<** RefCounted **<** Object

There is currently no description for this class. Please help us by contributing one!

Tutorials

- Runtime file loading and saving

Properties

Skin	godot_skin	
PackedInt32Array	joints	`PackedInt32Array()`
PackedInt32Array	joints_original	`PackedInt32Array()`
PackedInt32Array	non_joints	`PackedInt32Array()`
PackedInt32Array	roots	`PackedInt32Array()`
int	skeleton	`-1`
int	skin_root	`-1`

Methods

Array[Transform3D]	get_inverse_binds()
Dictionary	get_joint_i_to_bone_i()
Dictionary	get_joint_i_to_name()
void	set_inverse_binds(inverse_binds: Array[Transform3D])
void	set_joint_i_to_bone_i(joint_i_to_bone_i: Dictionary)
void	set_joint_i_to_name(joint_i_to_name: Dictionary)

Property Descriptions

Skin **godot_skin** 🔗

- void **set_godot_skin**(value: Skin)
- Skin **get_godot_skin**()

There is currently no description for this property. Please help us by contributing one!

PackedInt32Array **joints** = `PackedInt32Array()`

- void **set_joints**(value: PackedInt32Array)
- PackedInt32Array **get_joints**()

There is currently no description for this property. Please help us by contributing one!

PackedInt32Array **joints_original** = `PackedInt32Array()`

- void **set_joints_original**(value: PackedInt32Array)
- PackedInt32Array **get_joints_original**()

There is currently no description for this property. Please help us by contributing one!

PackedInt32Array **non_joints** = `PackedInt32Array()`

- void **set_non_joints**(value: PackedInt32Array)
- PackedInt32Array **get_non_joints**()

There is currently no description for this property. Please help us by contributing one!

PackedInt32Array **roots** = `PackedInt32Array()`

- void **set_roots**(value: PackedInt32Array)
- PackedInt32Array **get_roots**()

There is currently no description for this property. Please help us by contributing one!

---

int **skeleton** = `-1`

- void **set_skeleton**(value: int)
- int **get_skeleton**()

There is currently no description for this property. Please help us by contributing one!

---

int **skin_root** = `-1`

- void **set_skin_root**(value: int)
- int **get_skin_root**()

There is currently no description for this property. Please help us by contributing one!

Method Descriptions

Array[Transform3D] **get_inverse_binds**()

There is currently no description for this method. Please help us by contributing one!

Dictionary **get_joint_i_to_bone_i**()

There is currently no description for this method. Please help us by contributing one!

Dictionary **get_joint_i_to_name**()

There is currently no description for this method. Please help us by contributing one!

---

void **set_inverse_binds**(inverse_binds: Array[Transform3D]) 🔗

There is currently no description for this method. Please help us by contributing one!

---

void **set_joint_i_to_bone_i**(joint_i_to_bone_i: Dictionary) 🔗

There is currently no description for this method. Please help us by contributing one!

---

void **set_joint_i_to_name**(joint_i_to_name: Dictionary) 🔗

There is currently no description for this method. Please help us by contributing one!

GLTFSpecGloss

Inherits: Resource **<** RefCounted **<** Object

Archived GLTF extension for specular/glossy materials.

Description

KHR_materials_pbrSpecularGlossiness is an archived GLTF extension. This means that it is deprecated and not recommended for new files. However, it is still supported for loading old files.

Tutorials

- Runtime file loading and saving
- KHR_materials_pbrSpecularGlossiness GLTF extension spec [https://github.com/KhronosGroup/glTF/blob/main/extensions/2.0/Archived/KHR_materials_pbrSpecularGlossiness]

Properties

Color	diffuse_factor	`Color(1, 1, 1, 1)`
Image	diffuse_img	
float	gloss_factor	`1.0`

Image	spec_gloss_img	
Color	specular_factor	`Color(1, 1, 1, 1)`

Property Descriptions

Color **diffuse_factor** = `Color(1, 1, 1, 1)`

- void **set_diffuse_factor**(value: Color)
- Color **get_diffuse_factor**()

The reflected diffuse factor of the material.

Image **diffuse_img**

- void **set_diffuse_img**(value: Image)
- Image **get_diffuse_img**()

The diffuse texture.

---

float **gloss_factor** = `1.0`

- void **set_gloss_factor**(value: float)
- float **get_gloss_factor**()

The glossiness or smoothness of the material.

Image **spec_gloss_img**

- void **set_spec_gloss_img**(value: Image)
- Image **get_spec_gloss_img**()

The specular-glossiness texture.

Color **specular_factor** = `Color(1, 1, 1, 1)` 🔗

- void **set_specular_factor**(value: Color)
- Color **get_specular_factor**()

The specular RGB color of the material. The alpha channel is unused.

GLTFState

Inherits: Resource **<** RefCounted **<** Object

Inherited By: FBXState

Represents all data of a GLTF file.

Description

Contains all nodes and resources of a GLTF file. This is used by GLTFDocument as data storage, which allows GLTFDocument and all GLTFDocumentExtension classes to remain stateless.

GLTFState can be populated by GLTFDocument reading a file or by converting a Godot scene. Then the data can either be used to create a Godot scene or save to a GLTF file. The code that converts to/from a Godot scene can be intercepted at arbitrary points by GLTFDocumentExtension classes. This allows for custom data to be stored in the GLTF file or for custom data to be converted to/from Godot nodes.

Tutorials

- Runtime file loading and saving
- GLTF asset header schema [https://github.com/KhronosGroup/glTF/blob/main/specification/2.0/schema/asset.schema.json"]

Properties

float	bake_fps	`30.0`
String	base_path	`""`
Array[PackedByteArray]	buffers	`[]`
String	copyright	`""`

bool	create_animations	`true`
String	filename	`""`
PackedByteArray	glb_data	`PackedByteArray()`
bool	import_as_skeleton_bones	`false`
Dictionary	json	`{}`
int	major_version	`0`
int	minor_version	`0`
PackedInt32Array	root_nodes	`PackedInt32Array()`
String	scene_name	`""`
bool	use_named_skin_binds	`false`

Methods

void	add_used_extension(extension_name: String, required: bool)
int	append_data_to_buffers(data: PackedByteArray, deduplication: bool)
Array[GLTFAccessor]	get_accessors()
Variant	get_additional_data(extension_name: StringName)

AnimationPlayer	get_animation_player(idx: int)
int	get_animation_players_count(idx: int)
Array[GLTFAnimation]	get_animations()
Array[GLTFBufferView]	get_buffer_views()
Array[GLTFCamera]	get_cameras()
int	get_handle_binary_image()
Array[Texture2D]	get_images()
Array[GLTFLight]	get_lights()
Array[Material]	get_materials()
Array[GLTFMesh]	get_meshes()
int	get_node_index(scene_node: Node)
Array[GLTFNode]	get_nodes()
Node	get_scene_node(idx: int)
Array[GLTFSkeleton]	get_skeletons()
Array[GLTFSkin]	get_skins()
Array[GLTFTextureSampler]	get_texture_samplers()

Array[GLTFTexture]	get_textures()
Array[String]	get_unique_animation_names()
Array[String]	get_unique_names()
void	set_accessors(accessors: Array[GLTFAccessor])
void	set_additional_data(extension_name: StringName, additional_data: Variant)
void	set_animations(animations: Array[GLTFAnimation])
void	set_buffer_views(buffer_views: Array[GLTFBufferView])
void	set_cameras(cameras: Array[GLTFCamera])
void	set_handle_binary_image(method: int)
void	set_images(images: Array[Texture2D])
void	set_lights(lights: Array[GLTFLight])
void	set_materials(materials: Array[Material])
void	set_meshes(meshes: Array[GLTFMesh])
void	set_nodes(nodes: Array[GLTFNode])
void	set_skeletons(skeletons: Array[GLTFSkeleton])
void	set_skins(skins: Array[GLTFSkin])

void	set_texture_samplers(texture_samplers: Array[GLTFTextureSampler])
void	set_textures(textures: Array[GLTFTexture])
void	set_unique_animation_names(unique_animation_names: Array[String])
void	set_unique_names(unique_names: Array[String])

Constants

HANDLE_BINARY_DISCARD_TEXTURES = `0`

Discards all embedded textures and uses untextured materials.

HANDLE_BINARY_EXTRACT_TEXTURES = `1`

Extracts embedded textures to be reimported and compressed. Editor only. Acts as uncompressed at runtime.

HANDLE_BINARY_EMBED_AS_BASISU = `2`

Embeds textures VRAM compressed with Basis Universal into the generated scene.

HANDLE_BINARY_EMBED_AS_UNCOMPRESSED = `3`

Embeds textures compressed losslessly into the generated scene, matching old behavior.

Property Descriptions

float **bake_fps** = `30.0`

- void **set_bake_fps**(value: float)
- float **get_bake_fps**()

The baking fps of the animation for either import or export.

String **base_path** = `""`

- void **set_base_path**(value: String)
- String **get_base_path**()

The folder path associated with this GLTF data. This is used to find other files the GLTF file references, like images or binary buffers. This will be set during import when appending from a file, and will be set during export when writing to a file.

Array[PackedByteArray] **buffers** = `[]`

- void **set_buffers**(value: Array[PackedByteArray])
- Array[PackedByteArray] **get_buffers**()

There is currently no description for this property. Please help us by contributing one!

String **copyright** = `""`

- void **set_copyright**(value: String)
- String **get_copyright**()

The copyright string in the asset header of the GLTF file. This is set during import if present and export if non-empty. See the GLTF asset header documentation for more information.

---

bool **create_animations** = `true`

- void **set_create_animations**(value: bool)
- bool **get_create_animations**()

There is currently no description for this property. Please help us by contributing one!

String **filename** = `""`

- void **set_filename**(value: String)
- String **get_filename**()

The file name associated with this GLTF data. If it ends with `.gltf`, this is text-based GLTF, otherwise this is binary GLB. This will be set during import when appending from a file, and will be set during export when writing to a file. If writing to a buffer, this will be an empty string.

PackedByteArray **glb_data** = `PackedByteArray()`

- void **set_glb_data**(value: PackedByteArray)
- PackedByteArray **get_glb_data**()

The binary buffer attached to a .glb file.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedByteArray for more details.

---

bool **import_as_skeleton_bones** = `false`

- void **set_import_as_skeleton_bones**(value: bool)
- bool **get_import_as_skeleton_bones**()

True to force all GLTFNodes in the document to be bones of a single Skeleton3D godot node.

Dictionary **json** = `{}`

- void **set_json**(value: Dictionary)
- Dictionary **get_json**()

The original raw JSON document corresponding to this GLTFState.

---

int **major_version** = `0`

- void **set_major_version**(value: int)
- int **get_major_version**()

There is currently no description for this property. Please help us by contributing one!

---

int **minor_version** = `0`

- void **set_minor_version**(value: int)
- int **get_minor_version**()

There is currently no description for this property. Please help us by contributing one!

PackedInt32Array **root_nodes** = `PackedInt32Array()`

- void **set_root_nodes**(value: PackedInt32Array)
- PackedInt32Array **get_root_nodes**()

The root nodes of the GLTF file. Typically, a GLTF file will only have one scene, and therefore one root node. However, a GLTF file may have multiple scenes and therefore multiple root nodes, which will be generated as siblings of each other and as children of the root node of the generated Godot scene.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedInt32Array for more details.

String **scene_name** = `""`

- void **set_scene_name**(value: String)
- String **get_scene_name**()

The name of the scene. When importing, if not specified, this will be the file name. When exporting, if specified, the scene name will be saved to the GLTF file.

bool **use_named_skin_binds** = `false`

- void **set_use_named_skin_binds**(value: bool)
- bool **get_use_named_skin_binds**()

There is currently no description for this property. Please help us by contributing one!

Method Descriptions

void **add_used_extension**(extension_name: String, required: bool)

Appends an extension to the list of extensions used by this GLTF file during serialization. If `required` is true, the extension will also be added to the list of required extensions. Do not run this in GLTFDocumentExtension._export_post, as that stage is too late to add extensions. The final list is sorted alphabetically.

int **append_data_to_buffers**(data: PackedByteArray, deduplication: bool)

Appends the given byte array data to the buffers and creates a GLTFBufferView for it. The index of the destination GLTFBufferView is returned. If `deduplication` is true, the buffers will first be searched for duplicate data, otherwise new bytes will always be appended.

Array[GLTFAccessor] **get_accessors**()

There is currently no description for this method. Please help us by contributing one!

Variant **get_additional_data**(extension_name: StringName)

Gets additional arbitrary data in this **GLTFState** instance. This can be used to keep per-file state data in GLTFDocumentExtension classes, which is important because they are stateless.

The argument should be the GLTFDocumentExtension name (does not have to match the extension name in the GLTF file), and the return value can be anything you set. If nothing was set, the return value is null.

AnimationPlayer **get_animation_player**(idx: int)

Returns the AnimationPlayer node with the given index. These nodes are only used during the export process when converting Godot AnimationPlayer nodes to GLTF animations.

---

int **get_animation_players_count**(idx: int)

Returns the number of AnimationPlayer nodes in this **GLTFState**. These nodes are only used during the export process when converting Godot AnimationPlayer nodes to GLTF animations.

Array[GLTFAnimation] **get_animations**()

Returns an array of all GLTFAnimations in the GLTF file. When importing, these will be generated as animations in an AnimationPlayer node. When exporting, these will be generated from Godot AnimationPlayer nodes.

Array[GLTFBufferView] **get_buffer_views**()

There is currently no description for this method. Please help us by contributing one!

Array[GLTFCamera] **get_cameras**()

Returns an array of all GLTFCameras in the GLTF file. These are the cameras that the GLTFNode.camera index refers to.

---

int **get_handle_binary_image**()

There is currently no description for this method. Please help us by contributing one!

Array[Texture2D] **get_images**()

Gets the images of the GLTF file as an array of Texture2Ds. These are the images that the GLTFTexture.src_image index refers to.

Array[GLTFLight] **get_lights**()

Returns an array of all GLTFLights in the GLTF file. These are the lights that the GLTFNode.light index refers to.

Array[Material] **get_materials**()

There is currently no description for this method. Please help us by contributing one!

Array[GLTFMesh] **get_meshes**()

Returns an array of all GLTFMeshes in the GLTF file. These are the meshes that the GLTFNode.mesh index refers to.

---

int **get_node_index**(scene_node: Node)

Returns the index of the GLTFNode corresponding to this Godot scene node. This is the inverse of get_scene_node. Useful during the export process.

Note: Not every Godot scene node will have a corresponding GLTFNode, and not every GLTFNode will have a scene node generated. If there is no GLTFNode index for this scene node, `-1` is returned.

Array[GLTFNode] **get_nodes**()

Returns an array of all GLTFNodes in the GLTF file. These are the nodes that GLTFNode.children and root_nodes refer to. This includes nodes that may not be generated in the Godot scene, or nodes that may generate multiple Godot scene nodes.

Node **get_scene_node**(idx: int) 🔗

Returns the Godot scene node that corresponds to the same index as the GLTFNode it was generated from. This is the inverse of get_node_index. Useful during the import process.

Note: Not every GLTFNode will have a scene node generated, and not every generated scene node will have a corresponding GLTFNode. If there is no scene node for this GLTFNode index, `null` is returned.

Array[GLTFSkeleton] **get_skeletons**() 🔗

Returns an array of all GLTFSkeletons in the GLTF file. These are the skeletons that the GLTFNode.skeleton index refers to.

Array[GLTFSkin] **get_skins**() 🔗

Returns an array of all GLTFSkins in the GLTF file. These are the skins that the GLTFNode.skin index refers to.

Array[GLTFTextureSampler] **get_texture_samplers**() 🔗

Retrieves the array of texture samplers that are used by the textures contained in the GLTF.

Array[GLTFTexture] **get_textures**() 🔗

There is currently no description for this method. Please help us by contributing one!

Array[String] **get_unique_animation_names**() 🔗

Returns an array of unique animation names. This is only used during the import process.

Array[String] **get_unique_names**()

Returns an array of unique node names. This is used in both the import process and export process.

---

void **set_accessors**(accessors: Array[GLTFAccessor])

There is currently no description for this method. Please help us by contributing one!

---

void **set_additional_data**(extension_name: StringName, additional_data: Variant)

Sets additional arbitrary data in this **GLTFState** instance. This can be used to keep per-file state data in GLTFDocumentExtension classes, which is important because they are stateless.

The first argument should be the GLTFDocumentExtension name (does not have to match the extension name in the GLTF file), and the second argument can be anything you want.

---

void **set_animations**(animations: Array[GLTFAnimation])

Sets the GLTFAnimations in the state. When importing, these will be generated as animations in an AnimationPlayer node. When exporting, these will be generated from Godot AnimationPlayer nodes.

---

void **set_buffer_views**(buffer_views: Array[GLTFBufferView])

There is currently no description for this method. Please help us by contributing one!

---

void **set_cameras**(cameras: Array[GLTFCamera])

Sets the GLTFCameras in the state. These are the cameras that the GLTFNode.camera index refers to.

---

void **set_handle_binary_image**(method: int)

There is currently no description for this method. Please help us by contributing one!

---

void **set_images**(images: Array[Texture2D]) 🔗

Sets the images in the state stored as an array of Texture2Ds. This can be used during export. These are the images that the GLTFTexture.src_image index refers to.

---

void **set_lights**(lights: Array[GLTFLight]) 🔗

Sets the GLTFLights in the state. These are the lights that the GLTFNode.light index refers to.

---

void **set_materials**(materials: Array[Material]) 🔗

There is currently no description for this method. Please help us by contributing one!

---

void **set_meshes**(meshes: Array[GLTFMesh]) 🔗

Sets the GLTFMeshes in the state. These are the meshes that the GLTFNode.mesh index refers to.

---

void **set_nodes**(nodes: Array[GLTFNode]) 🔗

Sets the GLTFNodes in the state. These are the nodes that GLTFNode.children and root_nodes refer to. Some of the nodes set here may not be generated in the Godot scene, or may generate multiple Godot scene nodes.

---

void **set_skeletons**(skeletons: Array[GLTFSkeleton]) 🔗

Sets the GLTFSkeletons in the state. These are the skeletons that the GLTFNode.skeleton index refers to.

---

void **set_skins**(skins: Array[GLTFSkin]) 🔗

Sets the GLTFSkins in the state. These are the skins that the GLTFNode.skin index refers to.

---

void **set_texture_samplers**(texture_samplers: Array[GLTFTextureSampler]) 🔗

Sets the array of texture samplers that are used by the textures contained in the GLTF.

---

void **set_textures**(textures: Array[GLTFTexture])

There is currently no description for this method. Please help us by contributing one!

---

void **set_unique_animation_names**(unique_animation_names: Array[String])

Sets the unique animation names in the state. This is only used during the import process.

---

void **set_unique_names**(unique_names: Array[String])

Sets the unique node names in the state. This is used in both the import process and export process.

GLTFTexture

Inherits: Resource **<** RefCounted **<** Object

GLTFTexture represents a texture in a GLTF file.

Tutorials

- Runtime file loading and saving

Properties

int	sampler	`-1`
int	src_image	`-1`

Property Descriptions

int **sampler** = `-1` 🔗

- void **set_sampler**(value: int)
- int **get_sampler**()

ID of the texture sampler to use when sampling the image. If -1, then the default texture sampler is used (linear filtering, and repeat wrapping in both axes).

---

int **src_image** = `-1` 🔗

- void **set_src_image**(value: int)
- int **get_src_image**()

The index of the image associated with this texture, see GLTFState.get_images. If -1, then this texture does not have an image assigned.

GLTFTextureSampler

Inherits: Resource **<** RefCounted **<** Object

Represents a GLTF texture sampler

Description

Represents a texture sampler as defined by the base GLTF spec. Texture samplers in GLTF specify how to sample data from the texture's base image, when rendering the texture on an object.

Tutorials

- Runtime file loading and saving

Properties

int	mag_filter	`9729`
int	min_filter	`9987`
int	wrap_s	`10497`
int	wrap_t	`10497`

Property Descriptions

int **mag_filter** = `9729`

- void **set_mag_filter**(value: int)
- int **get_mag_filter**()

Texture's magnification filter, used when texture appears larger on screen than the source image.

---

int **min_filter** = `9987`

- void **set_min_filter**(value: int)
- int **get_min_filter**()

Texture's minification filter, used when the texture appears smaller on screen than the source image.

---

int **wrap_s** = `10497`

- void **set_wrap_s**(value: int)
- int **get_wrap_s**()

Wrapping mode to use for S-axis (horizontal) texture coordinates.

---

int **wrap_t** = `10497`

- void **set_wrap_t**(value: int)
- int **get_wrap_t**()

Wrapping mode to use for T-axis (vertical) texture coordinates.

Gradient

Inherits: Resource **<** RefCounted **<** Object

A color transition.

Description

This resource describes a color transition by defining a set of colored points and how to interpolate between them.

See also Curve which supports more complex easing methods, but does not support colors.

Properties

PackedColorArray	colors	`PackedColorArray(0, 0, 0, 1, 1, 1, 1, 1)`
ColorSpace	interpolation_color_space	`0`
InterpolationMode	interpolation_mode	`0`
PackedFloat32Array	offsets	`PackedFloat32Array(0, 1)`

Methods

void	add_point(offset: float, color: Color)
Color	get_color(point: int)
float	get_offset(point: int)
int	get_point_count() const
void	remove_point(point: int)
void	reverse()
Color	sample(offset: float)
void	set_color(point: int, color: Color)
void	set_offset(point: int, offset: float)

Enumerations

enum **InterpolationMode**: 🔗

InterpolationMode **GRADIENT_INTERPOLATE_LINEAR** = `0`

Linear interpolation.

InterpolationMode **GRADIENT_INTERPOLATE_CONSTANT** = `1`

Constant interpolation, color changes abruptly at each point and stays uniform between. This might cause visible aliasing when used for a gradient texture in some cases.

InterpolationMode **GRADIENT_INTERPOLATE_CUBIC** = `2`

Cubic interpolation.

---

enum **ColorSpace**:

ColorSpace **GRADIENT_COLOR_SPACE_SRGB** = `0`

sRGB color space.

ColorSpace **GRADIENT_COLOR_SPACE_LINEAR_SRGB** = `1`

Linear sRGB color space.

ColorSpace **GRADIENT_COLOR_SPACE_OKLAB** = `2`

Oklab [https://bottosson.github.io/posts/oklab/] color space. This color space provides a smooth and uniform-looking transition between colors.

Property Descriptions

PackedColorArray **colors** = `PackedColorArray(0, 0, 0, 1, 1, 1, 1, 1)`

- void **set_colors**(value: PackedColorArray)
- PackedColorArray **get_colors**()

Gradient's colors as a PackedColorArray.

Note: Setting this property updates all colors at once. To update any color individually use set_color.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedColorArray for more details.

ColorSpace **interpolation_color_space** = `0` 🔗

- void **set_interpolation_color_space**(value: ColorSpace)
- ColorSpace **get_interpolation_color_space**()

The color space used to interpolate between points of the gradient. It does not affect the returned colors, which will always be in sRGB space. See ColorSpace for available modes.

Note: This setting has no effect when interpolation_mode is set to GRADIENT_INTERPOLATE_CONSTANT.

InterpolationMode **interpolation_mode** = `0` 🔗

- void **set_interpolation_mode**(value: InterpolationMode)
- InterpolationMode **get_interpolation_mode**()

The algorithm used to interpolate between points of the gradient. See InterpolationMode for available modes.

PackedFloat32Array **offsets** = `PackedFloat32Array(0, 1)` 🔗

- void **set_offsets**(value: PackedFloat32Array)
- PackedFloat32Array **get_offsets**()

Gradient's offsets as a PackedFloat32Array.

Note: Setting this property updates all offsets at once. To update any offset individually use set_offset.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedFloat32Array for more details.

Method Descriptions

void **add_point**(offset: float, color: Color)

Adds the specified color to the gradient, with the specified offset.

Color **get_color**(point: int)

Returns the color of the gradient color at index `point`.

---

float **get_offset**(point: int)

Returns the offset of the gradient color at index `point`.

---

int **get_point_count**() const

Returns the number of colors in the gradient.

---

void **remove_point**(point: int)

Removes the color at index `point`.

---

void **reverse**()

Reverses/mirrors the gradient.

Note: This method mirrors all points around the middle of the gradient, which may produce unexpected results when interpolation_mode is set to GRADIENT_INTERPOLATE_CONSTANT.

Color **sample**(offset: float)

Returns the interpolated color specified by `offset`.

---

void **set_color**(point: int, color: Color)

Sets the color of the gradient color at index `point`.

---

void **set_offset**(point: int, offset: float)

Sets the offset for the gradient color at index `point`.

GradientTexture1D

Inherits: Texture2D **<** Texture **<** Resource **<** RefCounted **<** Object

A 1D texture that uses colors obtained from a Gradient.

Description

A 1D texture that obtains colors from a Gradient to fill the texture data. The texture is filled by sampling the gradient for each pixel. Therefore, the texture does not necessarily represent an exact copy of the gradient, as it may miss some colors if there are not enough pixels. See also GradientTexture2D, CurveTexture and CurveXYZTexture.

Properties

Gradient	gradient	
bool	resource_local_to_scene	`false` (overrides Resource)
bool	use_hdr	`false`
int	width	`256`

Property Descriptions

Gradient **gradient** 🔗

- void **set_gradient**(value: Gradient)
- Gradient **get_gradient**()

The Gradient used to fill the texture.

---

bool **use_hdr** = `false` 🔗

- void **set_use_hdr**(value: bool)
- bool **is_using_hdr**()

If `true`, the generated texture will support high dynamic range (Image.FORMAT_RGBAF format). This allows for glow effects to work if Environment.glow_enabled is `true`. If `false`, the generated texture will use low dynamic range; overbright colors will be clamped (Image.FORMAT_RGBA8 format).

---

int **width** = `256` 🔗

- void **set_width**(value: int)
- int **get_width**()

The number of color samples that will be obtained from the Gradient.

GradientTexture2D

Inherits: Texture2D **<** Texture **<** Resource **<** RefCounted **<** Object

A 2D texture that creates a pattern with colors obtained from a Gradient.

Description

A 2D texture that obtains colors from a Gradient to fill the texture data. This texture is able to transform a color transition into different patterns such as a linear or a radial gradient. The gradient is sampled individually for each pixel so it does not necessarily represent an exact copy of the gradient(see width and height). See also GradientTexture1D, CurveTexture and CurveXYZTexture.

Properties

Fill	fill	`0`
Vector2	fill_from	`Vector2(0, 0)`
Vector2	fill_to	`Vector2(1, 0)`
Gradient	gradient	
int	height	`64`

Repeat	repeat	`0`
bool	resource_local_to_scene	`false` (overrides Resource)
bool	use_hdr	`false`
int	width	`64`

Enumerations

enum **Fill**: 🔗

Fill **FILL_LINEAR** = `0`

The colors are linearly interpolated in a straight line.

Fill **FILL_RADIAL** = `1`

The colors are linearly interpolated in a circular pattern.

Fill **FILL_SQUARE** = `2`

The colors are linearly interpolated in a square pattern.

enum **Repeat**: 🔗

Repeat **REPEAT_NONE** = `0`

The gradient fill is restricted to the range defined by fill_from to fill_to offsets.

Repeat **REPEAT** = `1`

The texture is filled starting from fill_from to fill_to offsets, repeating the same pattern in both directions.

Repeat **REPEAT_MIRROR** = `2`

The texture is filled starting from fill_from to fill_to offsets, mirroring the pattern in both directions.

Property Descriptions

Fill **fill** = `0`

- void **set_fill**(value: Fill)
- Fill **get_fill**()

The gradient fill type, one of the Fill values. The texture is filled by interpolating colors starting from fill_from to fill_to offsets.

Vector2 **fill_from** = `Vector2(0, 0)`

- void **set_fill_from**(value: Vector2)
- Vector2 **get_fill_from**()

The initial offset used to fill the texture specified in UV coordinates.

Vector2 **fill_to** = `Vector2(1, 0)`

- void **set_fill_to**(value: Vector2)
- Vector2 **get_fill_to**()

The final offset used to fill the texture specified in UV coordinates.

Gradient **gradient**

- void **set_gradient**(value: Gradient)
- Gradient **get_gradient**()

The Gradient used to fill the texture.

---

int **height** = `64`

- void **set_height**(value: int)
- int **get_height**()

The number of vertical color samples that will be obtained from the Gradient, which also represents the texture's height.

Repeat **repeat** = `0`

- void **set_repeat**(value: Repeat)
- Repeat **get_repeat**()

The gradient repeat type, one of the Repeat values. The texture is filled starting from fill_from to fill_to offsets by default, but the gradient fill can be repeated to cover the entire texture.

---

bool **use_hdr** = `false`

- void **set_use_hdr**(value: bool)
- bool **is_using_hdr**()

If `true`, the generated texture will support high dynamic range (Image.FORMAT_RGBAF format). This allows for glow effects to work if Environment.glow_enabled is `true`. If `false`, the generated texture will use low dynamic range; overbright colors will be clamped (Image.FORMAT_RGBA8 format).

---

int **width** = `64` 🔗

- void **set_width**(value: int)
- int **get_width**()

The number of horizontal color samples that will be obtained from the Gradient, which also represents the texture's width.

HeightMapShape3D

Inherits: Shape3D **<** Resource **<** RefCounted **<** Object

A 3D height map shape used for physics collision.

Description

A 3D heightmap shape, intended for use in physics. Usually used to provide a shape for a CollisionShape3D. This is useful for terrain, but it is limited as overhangs (such as caves) cannot be stored. Holes in a **HeightMapShape3D** are created by assigning very low values to points in the desired area.

Performance: HeightMapShape3D is faster to check collisions against than ConcavePolygonShape3D, but it is significantly slower than primitive shapes like BoxShape3D.

A heightmap collision shape can also be build by using an Image reference:

GDScript

```
var heightmap_texture: Texture =
ResourceLoader.load("res://heightmap_image.exr")
var heightmap_image: Image = heightmap_texture.get_image()
heightmap_image.convert(Image.FORMAT_RF)

var height_min: float = 0.0
var height_max: float = 10.0

update_map_data_from_image(heightmap_image, height_min,
height_max)
```

Properties

PackedFloat32Array	map_data	`PackedFloat32Array(0, 0, 0, 0)`
int	map_depth	`2`
int	map_width	`2`

Methods

float	get_max_height() const
float	get_min_height() const
void	update_map_data_from_image(image: Image, height_min: float, height_max: float)

Property Descriptions

PackedFloat32Array **map_data** = `PackedFloat32Array(0, 0, 0, 0)` 🔗

- void **set_map_data**(value: PackedFloat32Array)
- PackedFloat32Array **get_map_data**()

Height map data. The array's size must be equal to map_width multiplied by map_depth.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedFloat32Array for more details.

---

int **map_depth** = `2` 🔗

- void **set_map_depth**(value: int)
- int **get_map_depth**()

Number of vertices in the depth of the height map. Changing this will resize the map_data.

---

int **map_width** = `2` 🔗

- void **set_map_width**(value: int)
- int **get_map_width**()

Number of vertices in the width of the height map. Changing this will resize the map_data.

Method Descriptions

float **get_max_height**() const 🔗

Returns the largest height value found in map_data. Recalculates only when map_data changes.

---

float **get_min_height**() const 🔗

Returns the smallest height value found in map_data. Recalculates only when map_data changes.

---

void **update_map_data_from_image**(image: Image, height_min: float, height_max: float) 🔗

Updates map_data with data read from an Image reference. Automatically resizes heightmap map_width and map_depth to fit the full image width and height.

The image needs to be in either Image.FORMAT_RF (32 bit), Image.FORMAT_RH (16 bit), or Image.FORMAT_R8 (8 bit).

Each image pixel is read in as a float on the range from `0.0` (black pixel) to `1.0` (white pixel). This range value gets remapped to `height_min` and `height_max` to form the final height value.

Image

Inherits: Resource **<** RefCounted **<** Object

Image datatype.

Description

Native image datatype. Contains image data which can be converted to an ImageTexture and provides commonly used *image processing* methods. The maximum width and height for an **Image** are MAX_WIDTH and MAX_HEIGHT.

An **Image** cannot be assigned to a texture property of an object directly (such as Sprite2D.texture), and has to be converted manually to an ImageTexture first.

Note: The maximum image size is 16384×16384 pixels due to graphics hardware limitations. Larger images may fail to import.

Tutorials

- Importing images
- Runtime file loading and saving

Properties

Dictionary	data	`{ "data": PackedByteArray(), "format": "Lum8", "height": 0, "mipmaps": false, "width": 0 }`

Methods

void	adjust_bcs(brightness: float, contrast: float, saturation: float)
void	blend_rect(src: Image, src_rect: Rect2i, dst: Vector2i)
void	blend_rect_mask(src: Image, mask: Image, src_rect: Rect2i, dst: Vector2i)
void	blit_rect(src: Image, src_rect: Rect2i, dst: Vector2i)
void	blit_rect_mask(src: Image, mask: Image, src_rect: Rect2i, dst: Vector2i)
void	bump_map_to_normal_map(bump_scale: float = 1.0)
void	clear_mipmaps()
Error	compress(mode: CompressMode, source: CompressSource = 0, astc_format: ASTCFormat = 0)

Error	compress_from_channels(mode: CompressMode, channels: UsedChannels, astc_format: ASTCFormat = 0)
Dictionary	compute_image_metrics(compared_image: Image, use_luma: bool)
void	convert(format: Format)
void	copy_from(src: Image)
Image	create(width: int, height: int, use_mipmaps: bool, format: Format) static
Image	create_empty(width: int, height: int, use_mipmaps: bool, format: Format) static
Image	create_from_data(width: int, height: int, use_mipmaps: bool, format: Format, data: PackedByteArray) static
void	crop(width: int, height: int)
Error	decompress()
AlphaMode	detect_alpha() const

UsedChannels	detect_used_channels(source: CompressSource = 0) const
void	fill(color: Color)
void	fill_rect(rect: Rect2i, color: Color)
void	fix_alpha_edges()
void	flip_x()
void	flip_y()
Error	generate_mipmaps(renormalize: bool = false)
PackedByteArray	get_data() const
int	get_data_size() const
Format	get_format() const
int	get_height() const
int	get_mipmap_count() const

int	get_mipmap_offset(mipmap: int) const
Color	get_pixel(x: int, y: int) const
Color	get_pixelv(point: Vector2i) const
Image	get_region(region: Rect2i) const
Vector2i	get_size() const
Rect2i	get_used_rect() const
int	get_width() const
bool	has_mipmaps() const
bool	is_compressed() const
bool	is_empty() const
bool	is_invisible() const
Error	load(path: String)
Error	load_bmp_from_buffer(buffer: PackedByteArray)

Image	load_from_file(path: String) static
Error	load_jpg_from_buffer(buffer: PackedByteArray)
Error	load_ktx_from_buffer(buffer: PackedByteArray)
Error	load_png_from_buffer(buffer: PackedByteArray)
Error	load_svg_from_buffer(buffer: PackedByteArray, scale: float = 1.0)
Error	load_svg_from_string(svg_str: String, scale: float = 1.0)
Error	load_tga_from_buffer(buffer: PackedByteArray)
Error	load_webp_from_buffer(buffer: PackedByteArray)
void	normal_map_to_xy()
void	premultiply_alpha()
void	resize(width: int, height: int, interpolation: Interpolation = 1)

void	resize_to_po2(square: bool = false, interpolation: Interpolation = 1)
Image	rgbe_to_srgb()
void	rotate_90(direction: ClockDirection)
void	rotate_180()
Error	save_exr(path: String, grayscale: bool = false) const
PackedByteArray	save_exr_to_buffer(grayscale: bool = false) const
Error	save_jpg(path: String, quality: float = 0.75) const
PackedByteArray	save_jpg_to_buffer(quality: float = 0.75) const
Error	save_png(path: String) const
PackedByteArray	save_png_to_buffer() const
Error	save_webp(path: String, lossy: bool = false, quality: float = 0.75) const

PackedByteArray	save_webp_to_buffer(lossy: bool = false, quality: float = 0.75) const
void	set_data(width: int, height: int, use_mipmaps: bool, format: Format, data: PackedByteArray)
void	set_pixel(x: int, y: int, color: Color)
void	set_pixelv(point: Vector2i, color: Color)
void	shrink_x2()
void	srgb_to_linear()

Enumerations

enum **Format**: 🔗

Format **FORMAT_L8** = `0`

Texture format with a single 8-bit depth representing luminance.

Format **FORMAT_LA8** = `1`

OpenGL texture format with two values, luminance and alpha each stored with 8 bits.

Format **FORMAT_R8** = `2`

OpenGL texture format `RED` with a single component and a bitdepth of 8.

Format **FORMAT_RG8** = `3`

OpenGL texture format `RG` with two components and a bitdepth of 8 for each.

Format **FORMAT_RGB8** = `4`

OpenGL texture format `RGB` with three components, each with a bitdepth of 8.

Note: When creating an ImageTexture, an sRGB to linear color space conversion is performed.

Format **FORMAT_RGBA8** = `5`

OpenGL texture format `RGBA` with four components, each with a bitdepth of 8.

Note: When creating an ImageTexture, an sRGB to linear color space conversion is performed.

Format **FORMAT_RGBA4444** = `6`

OpenGL texture format `RGBA` with four components, each with a bitdepth of 4.

Format **FORMAT_RGB565** = `7`

OpenGL texture format `RGB` with three components. Red and blue have a bitdepth of 5, and green has a bitdepth of 6.

Format **FORMAT_RF** = `8`

OpenGL texture format `GL_R32F` where there's one component, a 32-bit floating-point value.

Format **FORMAT_RGF** = `9`

OpenGL texture format `GL_RG32F` where there are two components, each a 32-bit floating-point values.

Format **FORMAT_RGBF** = `10`

OpenGL texture format `GL_RGB32F` where there are three components, each a 32-bit floating-point values.

Format **FORMAT_RGBAF** = `11`

OpenGL texture format `GL_RGBA32F` where there are four components, each a 32-bit floating-point values.

Format **FORMAT_RH** = `12`

OpenGL texture format `GL_R16F` where there's one component, a 16-bit "half-precision" floating-point value.

Format **FORMAT_RGH** = `13`

OpenGL texture format `GL_RG16F` where there are two components, each a 16-bit "half-precision" floating-point value.

Format **FORMAT_RGBH** = `14`

OpenGL texture format `GL_RGB16F` where there are three components, each a 16-bit "half-precision" floating-point value.

Format **FORMAT_RGBAH** = `15`

OpenGL texture format `GL_RGBA16F` where there are four components, each a 16-bit "half-precision" floating-point value.

Format **FORMAT_RGBE9995** = `16`

A special OpenGL texture format where the three color components have 9 bits of precision and all three share a single 5-bit exponent.

Format **FORMAT_DXT1** = `17`

The S3TC [https://en.wikipedia.org/wiki/S3_Texture_Compression] texture format that uses Block Compression 1, and is the smallest variation of S3TC, only providing 1 bit of alpha and color data being premultiplied with alpha.

Note: When creating an ImageTexture, an sRGB to linear color space conversion is performed.

Format **FORMAT_DXT3** = `18`

The S3TC [https://en.wikipedia.org/wiki/S3_Texture_Compression] texture format that uses Block Compression 2, and color data is interpreted as not having been premultiplied by alpha. Well suited for images with sharp alpha transitions between translucent and opaque areas.

Note: When creating an ImageTexture, an sRGB to linear color space conversion is performed.

Format **FORMAT_DXT5** = `19`

The S3TC [https://en.wikipedia.org/wiki/S3_Texture_Compression] texture format also known as Block Compression 3 or BC3 that contains 64 bits of alpha channel data followed by 64 bits of DXT1-encoded color data. Color data is not premultiplied by alpha, same as DXT3. DXT5 generally produces superior results for transparent gradients compared to DXT3.

Note: When creating an ImageTexture, an sRGB to linear color space conversion is performed.

Format **FORMAT_RGTC_R** = `20`

Texture format that uses Red Green Texture Compression [https://www.khronos.org/opengl/wiki/Red_Green_Texture_Compression], normalizing the red channel data using the same compression algorithm that DXT5 uses for the alpha channel.

Format **FORMAT_RGTC_RG** = `21`

Texture format that uses Red Green Texture Compression [https://www.khronos.org/opengl/wiki/Red_Green_Texture_Compression], normalizing the red and green channel data using the same compression algorithm that DXT5 uses for the alpha channel.

Format **FORMAT_BPTC_RGBA** = `22`

Texture format that uses BPTC [https://www.khronos.org/opengl/wiki/BPTC_Texture_Compression] compression with unsigned normalized RGBA components.

Note: When creating an ImageTexture, an sRGB to linear color space conversion is performed.

Format **FORMAT_BPTC_RGBF** = `23`

Texture format that uses BPTC [https://www.khronos.org/opengl/wiki/BPTC_Texture_Compression] compression with signed floating-point RGB components.

Format **FORMAT_BPTC_RGBFU** = `24`

Texture format that uses BPTC [https://www.khronos.org/opengl/wiki/BPTC_Texture_Compression] compression with unsigned floating-point RGB components.

Format **FORMAT_ETC** = `25`

Ericsson Texture Compression format 1 [https://en.wikipedia.org/wiki/Ericsson_Texture_Compression#ETC1], also referred to as "ETC1", and is part of the OpenGL ES graphics standard. This format cannot store an alpha channel.

Format **FORMAT_ETC2_R11** = `26`

Ericsson Texture Compression format 2 [https://en.wikipedia.org/wiki/Ericsson_Texture_Compression#ETC2_and_EAC] (`R11_EAC` variant), which provides one channel of unsigned data.

Format **FORMAT_ETC2_R11S** = `27`

Ericsson Texture Compression format 2 [https://en.wikipedia.org/wiki/Ericsson_Texture_Compression#ETC2_and_EAC] (`SIGNED_R11_EAC` variant), which provides one channel of signed data.

Format **FORMAT_ETC2_RG11** = `28`

Ericsson Texture Compression format 2 [https://en.wikipedia.org/wiki/Ericsson_Texture_Compression#ETC2_and_EAC] (`RG11_EAC` variant), which provides two channels of unsigned data.

Format **FORMAT_ETC2_RG11S** = `29`

Ericsson Texture Compression format 2 [https://en.wikipedia.org/wiki/Ericsson_Texture_Compression#ETC2_and_EAC] (`SIGNED_RG11_EAC` variant), which provides two channels of signed data.

Format **FORMAT_ETC2_RGB8** = `30`

Ericsson Texture Compression format 2 [https://en.wikipedia.org/wiki/Ericsson_Texture_Compression#ETC2_and_EAC] (`RGB8` variant), which is a follow-up of ETC1 and compresses RGB888 data.

Note: When creating an ImageTexture, an sRGB to linear color space conversion is performed.

Format **FORMAT_ETC2_RGBA8** = `31`

Ericsson Texture Compression format 2 [https://en.wikipedia.org/wiki/Ericsson_Texture_Compression#ETC2_and_EAC] (`RGBA8`variant), which compresses RGBA8888 data with full alpha support.

Note: When creating an ImageTexture, an sRGB to linear color space conversion is performed.

Format **FORMAT_ETC2_RGB8A1** = `32`

Ericsson Texture Compression format 2 [https://en.wikipedia.org/wiki/Ericsson_Texture_Compression#ETC2_and_EAC] (`RGB8_PUNCHTHROUGH_ALPHA1` variant), which compresses RGBA data to make alpha either fully transparent or fully opaque.

Note: When creating an ImageTexture, an sRGB to linear color space conversion is performed.

Format **FORMAT_ETC2_RA_AS_RG** = `33`

Ericsson Texture Compression format 2 [https://en.wikipedia.org/wiki/Ericsson_Texture_Compression#ETC2_and_EAC] (`RGBA8` variant), which compresses RA data and interprets it as two channels (red and green). See also FORMAT_ETC2_RGBA8.

Format **FORMAT_DXT5_RA_AS_RG** = `34`

The S3TC [https://en.wikipedia.org/wiki/S3_Texture_Compression] texture format also known as Block Compression 3 or BC3, which compresses RA data and interprets it as two channels (red and green). See also FORMAT_DXT5.

Format **FORMAT_ASTC_4x4** = `35`

Adaptive Scalable Texture Compression [https://en.wikipedia.org/wiki/Adaptive_scalable_texture_compression]. This implements the 4×4 (high quality) mode.

Format **FORMAT_ASTC_4x4_HDR** = `36`

Same format as FORMAT_ASTC_4x4, but with the hint to let the GPU know it is used for HDR.

Format **FORMAT_ASTC_8x8** = `37`

Adaptive Scalable Texture Compression [https://en.wikipedia.org/wiki/Adaptive_scalable_texture_compression]. This implements the 8×8 (low quality) mode.

Format **FORMAT_ASTC_8x8_HDR** = `38`

Same format as FORMAT_ASTC_8x8, but with the hint to let the GPU know it is used for HDR.

Format **FORMAT_MAX** = `39`

Represents the size of the Format enum.

---

enum **Interpolation**: 🔗

Interpolation **INTERPOLATE_NEAREST** = `0`

Performs nearest-neighbor interpolation. If the image is resized, it will be pixelated.

Interpolation **INTERPOLATE_BILINEAR** = `1`

Performs bilinear interpolation. If the image is resized, it will be blurry. This mode is faster than INTERPOLATE_CUBIC, but it results in lower quality.

Interpolation **INTERPOLATE_CUBIC** = `2`

Performs cubic interpolation. If the image is resized, it will be blurry. This mode often gives better results compared to INTERPOLATE_BILINEAR, at the cost of being slower.

Interpolation **INTERPOLATE_TRILINEAR** = `3`

Performs bilinear separately on the two most-suited mipmap levels, then linearly interpolates between them.

It's slower than INTERPOLATE_BILINEAR, but produces higher-quality results with far fewer aliasing artifacts.

If the image does not have mipmaps, they will be generated and used internally, but no mipmaps will be generated on the resulting image.

Note: If you intend to scale multiple copies of the original image, it's better to call generate_mipmaps] on it in advance, to avoid wasting processing power in generating them again and again.

On the other hand, if the image already has mipmaps, they will be used, and a new set will be generated for the resulting image.

Interpolation **INTERPOLATE_LANCZOS** = `4`

Performs Lanczos interpolation. This is the slowest image resizing mode, but it typically gives the best results, especially when downscaling images.

---

enum **AlphaMode**: 🔗

AlphaMode **ALPHA_NONE** = `0`

Image does not have alpha.

AlphaMode **ALPHA_BIT** = `1`

Image stores alpha in a single bit.

AlphaMode **ALPHA_BLEND** = `2`

Image uses alpha.

enum **CompressMode**: 🔗

CompressMode **COMPRESS_S3TC** = `0`

Use S3TC compression.

CompressMode **COMPRESS_ETC** = `1`

Use ETC compression.

CompressMode **COMPRESS_ETC2** = `2`

Use ETC2 compression.

CompressMode **COMPRESS_BPTC** = `3`

Use BPTC compression.

CompressMode **COMPRESS_ASTC** = `4`

Use ASTC compression.

CompressMode **COMPRESS_MAX** = `5`

Represents the size of the CompressMode enum.

---

enum **UsedChannels**: 🔗

UsedChannels **USED_CHANNELS_L** = `0`

The image only uses one channel for luminance (grayscale).

UsedChannels **USED_CHANNELS_LA** = `1`

The image uses two channels for luminance and alpha, respectively.

UsedChannels **USED_CHANNELS_R** = `2`

The image only uses the red channel.

UsedChannels **USED_CHANNELS_RG** = `3`

The image uses two channels for red and green.

UsedChannels **USED_CHANNELS_RGB** = `4`

The image uses three channels for red, green, and blue.

UsedChannels **USED_CHANNELS_RGBA** = `5`

The image uses four channels for red, green, blue, and alpha.

---

enum **CompressSource**: 🔗

CompressSource **COMPRESS_SOURCE_GENERIC** = `0`

Source texture (before compression) is a regular texture. Default for all textures.

CompressSource **COMPRESS_SOURCE_SRGB** = `1`

Source texture (before compression) is in sRGB space.

CompressSource **COMPRESS_SOURCE_NORMAL** = `2`

Source texture (before compression) is a normal texture (e.g. it can be compressed into two channels).

---

enum **ASTCFormat**:

ASTCFormat **ASTC_FORMAT_4x4** = `0`

Hint to indicate that the high quality 4×4 ASTC compression format should be used.

ASTCFormat **ASTC_FORMAT_8x8** = `1`

Hint to indicate that the low quality 8×8 ASTC compression format should be used.

Constants

MAX_WIDTH = `16777216`

The maximal width allowed for **Image** resources.

MAX_HEIGHT = `16777216`

The maximal height allowed for **Image** resources.

Property Descriptions

Dictionary **data** = `{ "data": PackedByteArray(), "format": "Lum8", "height": 0, "mipmaps": false, "width": 0 }`

Holds all the image's color data in a given format. See Format constants.

Method Descriptions

void **adjust_bcs**(brightness: float, contrast: float, saturation: float) 🔗

Adjusts this image's `brightness`, `contrast`, and `saturation` by the given values. Does not work if the image is compressed (see is_compressed).

void **blend_rect**(src: Image, src_rect: Rect2i, dst: Vector2i) 🔗

Alpha-blends `src_rect` from `src` image to this image at coordinates `dst`, clipped accordingly to both image bounds. This image and `src` image **must** have the same format. `src_rect` with non-positive size is treated as empty.

void **blend_rect_mask**(src: Image, mask: Image, src_rect: Rect2i, dst: Vector2i) 🔗

Alpha-blends `src_rect` from `src` image to this image using `mask` image at coordinates `dst`, clipped accordingly to both image bounds. Alpha channels are required for both `src` and `mask`. `dst` pixels and `src` pixels will blend if the corresponding mask pixel's alpha value is not 0. This image and `src` image **must** have the same format. `src` image and `mask` image **must** have the same size (width and height) but they can have different formats. `src_rect` with non-positive size is treated as empty.

void **blit_rect**(src: Image, src_rect: Rect2i, dst: Vector2i) 🔗

Copies `src_rect` from `src` image to this image at coordinates `dst`, clipped accordingly to both image bounds. This image and `src` image **must** have the same format. `src_rect` with non-positive size is treated as empty.

---

void **blit_rect_mask**(src: Image, mask: Image, src_rect: Rect2i, dst: Vector2i) 🔗

Blits `src_rect` area from `src` image to this image at the coordinates given by `dst`, clipped accordingly to both image bounds. `src` pixel is copied onto `dst` if the corresponding `mask` pixel's alpha value is not 0. This image and `src` image **must** have the same format. `src` image and `mask` image **must** have the same size (width and height) but they can have different formats. `src_rect` with non-positive size is treated as empty.

---

void **bump_map_to_normal_map**(bump_scale: float = 1.0) 🔗

Converts a bump map to a normal map. A bump map provides a height offset per-pixel, while a normal map provides a normal direction per pixel.

---

void **clear_mipmaps**() 🔗

Removes the image's mipmaps.

Error **compress**(mode: CompressMode, source: CompressSource = 0, astc_format: ASTCFormat = 0) 🔗

Compresses the image to use less memory. Can not directly access pixel data while the image is compressed. Returns error if the chosen compression mode is not available.

The `source` parameter helps to pick the best compression method for DXT and ETC2 formats. It is ignored for ASTC compression.

For ASTC compression, the `astc_format` parameter must be supplied.

Error **compress_from_channels**(mode: CompressMode, channels: UsedChannels, astc_format: ASTCFormat = 0)

Compresses the image to use less memory. Can not directly access pixel data while the image is compressed. Returns error if the chosen compression mode is not available.

This is an alternative to compress that lets the user supply the channels used in order for the compressor to pick the best DXT and ETC2 formats. For other formats (non DXT or ETC2), this argument is ignored.

For ASTC compression, the `astc_format` parameter must be supplied.

Dictionary **compute_image_metrics**(compared_image: Image, use_luma: bool)

Compute image metrics on the current image and the compared image.

The dictionary contains `max`, `mean`, `mean_squared`, `root_mean_squared` and `peak_snr`.

---

void **convert**(format: Format)

Converts the image's format. See Format constants.

---

void **copy_from**(src: Image)

Copies `src` image to this image.

Image **create**(width: int, height: int, use_mipmaps: bool, format: Format) static

Deprecated: Use create_empty.

Creates an empty image of given size and format. See Format constants. If `use_mipmaps` is `true`, then generate mipmaps for this image. See the generate_mipmaps.

Image **create_empty**(width: int, height: int, use_mipmaps: bool, format: Format) static

Creates an empty image of given size and format. See Format constants. If `use_mipmaps` is `true`, then generate mipmaps for this image. See the generate_mipmaps.

Image **create_from_data**(width: int, height: int, use_mipmaps: bool, format: Format, data: PackedByteArray) static

Creates a new image of given size and format. See Format constants. Fills the image with the given raw data. If `use_mipmaps` is `true` then loads mipmaps for this image from `data`. See generate_mipmaps.

---

void **crop**(width: int, height: int)

Crops the image to the given `width` and `height`. If the specified size is larger than the current size, the extra area is filled with black pixels.

Error **decompress**()

Decompresses the image if it is VRAM compressed in a supported format. Returns @GlobalScope.OK if the format is supported, otherwise @GlobalScope.ERR_UNAVAILABLE.

Note: The following formats can be decompressed: DXT, RGTC, BPTC. The formats ETC1 and ETC2 are not supported.

AlphaMode **detect_alpha**() const

Returns ALPHA_BLEND if the image has data for alpha values. Returns ALPHA_BIT if all the alpha values are stored in a single bit. Returns ALPHA_NONE if no data for alpha values is found.

UsedChannels **detect_used_channels**(source: CompressSource = 0) const

Returns the color channels used by this image, as one of the UsedChannels constants. If the image is compressed, the original `source` must be specified.

void **fill**(color: Color)

Fills the image with `color`.

void **fill_rect**(rect: Rect2i, color: Color)

Fills `rect` with `color`.

---

void **fix_alpha_edges**() 🔗

Blends low-alpha pixels with nearby pixels.

---

void **flip_x**() 🔗

Flips the image horizontally.

---

void **flip_y**() 🔗

Flips the image vertically.

Error **generate_mipmaps**(renormalize: bool = false) 🔗

Generates mipmaps for the image. Mipmaps are precalculated lower-resolution copies of the image that are automatically used if the image needs to be scaled down when rendered. They help improve image quality and performance when rendering. This method returns an error if the image is compressed, in a custom format, or if the image's width/height is `0`. Enabling `renormalize` when generating mipmaps for normal map textures will make sure all resulting vector values are normalized.

It is possible to check if the image has mipmaps by calling has_mipmaps or get_mipmap_count. Calling generate_mipmaps on an image that already has mipmaps will replace existing mipmaps in the image.

PackedByteArray **get_data**() const 🔗

Returns a copy of the image's raw data.

---

int **get_data_size**() const

Returns size (in bytes) of the image's raw data.

Format **get_format**() const

Returns the image's format. See Format constants.

---

int **get_height**() const

Returns the image's height.

---

int **get_mipmap_count**() const

Returns the number of mipmap levels or 0 if the image has no mipmaps. The largest main level image is not counted as a mipmap level by this method, so if you want to include it you can add 1 to this count.

---

int **get_mipmap_offset**(mipmap: int) const

Returns the offset where the image's mipmap with index `mipmap` is stored in the data dictionary.

Color **get_pixel**(x: int, y: int) const

Returns the color of the pixel at `(x, y)`.

This is the same as get_pixelv, but with two integer arguments instead of a Vector2i argument.

Color **get_pixelv**(point: Vector2i) const 🔗

Returns the color of the pixel at `point`.

This is the same as get_pixel, but with a Vector2i argument instead of two integer arguments.

Image **get_region**(region: Rect2i) const 🔗

Returns a new **Image** that is a copy of this **Image**'s area specified with `region`.

Vector2i **get_size**() const 🔗

Returns the image's size (width and height).

Rect2i **get_used_rect**() const 🔗

Returns a Rect2i enclosing the visible portion of the image, considering each pixel with a non-zero alpha channel as visible.

---

int **get_width**() const 🔗

Returns the image's width.

---

bool **has_mipmaps**() const 🔗

Returns `true` if the image has generated mipmaps.

---

bool **is_compressed**() const 🔗

Returns `true` if the image is compressed.

---

bool **is_empty**() const 🔗

Returns `true` if the image has no data.

---

bool **is_invisible**() const 🔗

Returns `true` if all the image's pixels have an alpha value of 0. Returns `false` if any pixel has an alpha value higher than 0.

Error **load**(path: String) 🔗

Loads an image from file `path`. See Supported image formats for a list of supported image formats and limitations.

Warning: This method should only be used in the editor or in cases when you need to load external images at run-time, such as images located at the `user://` directory, and may not work in exported projects.

See also ImageTexture description for usage examples.

Error **load_bmp_from_buffer**(buffer: PackedByteArray) 🔗

Loads an image from the binary contents of a BMP file.

Note: Godot's BMP module doesn't support 16-bit per pixel images. Only 1-bit, 4-bit, 8-bit, 24-bit, and 32-bit per pixel images are supported.

Note: This method is only available in engine builds with the BMP module enabled. By default, the BMP module is enabled, but it can be disabled at build-time using the `module_bmp_enabled=no` SCons option.

Image **load_from_file**(path: String) static

Creates a new **Image** and loads data from the specified file.

Error **load_jpg_from_buffer**(buffer: PackedByteArray)

Loads an image from the binary contents of a JPEG file.

Error **load_ktx_from_buffer**(buffer: PackedByteArray)

Loads an image from the binary contents of a KTX [https://github.com/KhronosGroup/KTX-Software] file. Unlike most image formats, KTX can store VRAM-compressed data and embed mipmaps.

Note: Godot's libktx implementation only supports 2D images. Cubemaps, texture arrays, and de-padding are not supported.

Note: This method is only available in engine builds with the KTX module enabled. By default, the KTX module is enabled, but it can be disabled at build-time using the `module_ktx_enabled=no` SCons option.

Error **load_png_from_buffer**(buffer: PackedByteArray)

Loads an image from the binary contents of a PNG file.

Error **load_svg_from_buffer**(buffer: PackedByteArray, scale: float = 1.0) 🔗

Loads an image from the UTF-8 binary contents of an **uncompressed** SVG file (**.svg**).

Note: Beware when using compressed SVG files (like **.svgz**), they need to be `decompressed` before loading.

Note: This method is only available in engine builds with the SVG module enabled. By default, the SVG module is enabled, but it can be disabled at build-time using the `module_svg_enabled=no` SCons option.

Error **load_svg_from_string**(svg_str: String, scale: float = 1.0) 🔗

Loads an image from the string contents of an SVG file (**.svg**).

Note: This method is only available in engine builds with the SVG module enabled. By default, the SVG module is enabled, but it can be disabled at build-time using the `module_svg_enabled=no` SCons option.

Error **load_tga_from_buffer**(buffer: PackedByteArray) 🔗

Loads an image from the binary contents of a TGA file.

Note: This method is only available in engine builds with the TGA module enabled. By default, the TGA module is enabled, but it can be disabled at build-time using the `module_tga_enabled=no` SCons option.

Error **load_webp_from_buffer**(buffer: PackedByteArray) 🔗

Loads an image from the binary contents of a WebP file.

---

void **normal_map_to_xy**()

Converts the image's data to represent coordinates on a 3D plane. This is used when the image represents a normal map. A normal map can add lots of detail to a 3D surface without increasing the polygon count.

---

void **premultiply_alpha**()

Multiplies color values with alpha values. Resulting color values for a pixel are `(color * alpha)/256`. See also CanvasItemMaterial.blend_mode.

---

void **resize**(width: int, height: int, interpolation: Interpolation = 1)

Resizes the image to the given `width` and `height`. New pixels are calculated using the `interpolation` mode defined via Interpolation constants.

---

void **resize_to_po2**(square: bool = false, interpolation: Interpolation = 1)

Resizes the image to the nearest power of 2 for the width and height. If `square` is `true` then set width and height to be the same. New pixels are calculated using the `interpolation` mode defined via Interpolation constants.

Image **rgbe_to_srgb**()

Converts a standard RGBE (Red Green Blue Exponent) image to an sRGB image.

void **rotate_90**(direction: ClockDirection)

Rotates the image in the specified `direction` by `90` degrees. The width and height of the image must be greater than `1`. If the width and height are not equal, the image will be resized.

void **rotate_180**()

Rotates the image by `180` degrees. The width and height of the image must be greater than `1`.

Error **save_exr**(path: String, grayscale: bool = false) const

Saves the image as an EXR file to `path`. If `grayscale` is `true` and the image has only one channel, it will be saved explicitly as monochrome rather than one red channel. This function will return @GlobalScope.ERR_UNAVAILABLE if Godot was compiled without the TinyEXR module.

Note: The TinyEXR module is disabled in non-editor builds, which means save_exr will return @GlobalScope.ERR_UNAVAILABLE when it is called from an exported project.

PackedByteArray **save_exr_to_buffer**(grayscale: bool = false) const

Saves the image as an EXR file to a byte array. If `grayscale` is `true` and the image has only one channel, it will be saved explicitly as monochrome rather than one red channel. This function will return an empty byte array if Godot was compiled without the TinyEXR module.

Note: The TinyEXR module is disabled in non-editor builds, which means save_exr will return an empty byte array when it is called from an exported project.

Error **save_jpg**(path: String, quality: float = 0.75) const

Saves the image as a JPEG file to `path` with the specified `quality` between `0.01` and `1.0` (inclusive). Higher `quality` values result in better-looking output at the cost of larger file sizes. Recommended `quality` values are between `0.75` and `0.90`. Even at quality `1.00`, JPEG compression remains lossy.

Note: JPEG does not save an alpha channel. If the **Image** contains an alpha channel, the image will still be saved, but the resulting JPEG file won't contain the alpha channel.

PackedByteArray **save_jpg_to_buffer**(quality: float = 0.75) const

Saves the image as a JPEG file to a byte array with the specified `quality` between `0.01` and `1.0` (inclusive). Higher `quality` values result in better-looking output at the cost of larger byte array sizes (and therefore memory usage). Recommended `quality` values are between `0.75` and `0.90`. Even at quality `1.00`, JPEG compression remains lossy.

Note: JPEG does not save an alpha channel. If the **Image** contains an alpha channel, the image will still be saved, but the resulting byte array won't contain the alpha channel.

Error **save_png**(path: String) const 🔗

Saves the image as a PNG file to the file at `path`.

PackedByteArray **save_png_to_buffer**() const 🔗

Saves the image as a PNG file to a byte array.

Error **save_webp**(path: String, lossy: bool = false, quality: float = 0.75) const 🔗

Saves the image as a WebP (Web Picture) file to the file at `path`. By default it will save lossless. If `lossy` is true, the image will be saved lossy, using the `quality` setting between 0.0 and 1.0 (inclusive). Lossless WebP offers more efficient compression than PNG.

Note: The WebP format is limited to a size of 16383×16383 pixels, while PNG can save larger images.

PackedByteArray **save_webp_to_buffer**(lossy: bool = false, quality: float = 0.75) const 🔗

Saves the image as a WebP (Web Picture) file to a byte array. By default it will save lossless. If `lossy` is true, the image will be saved lossy, using the `quality` setting between 0.0 and 1.0 (inclusive). Lossless WebP offers more efficient compression than PNG.

Note: The WebP format is limited to a size of 16383×16383 pixels, while PNG can save larger images.

---

void **set_data**(width: int, height: int, use_mipmaps: bool, format: Format, data: PackedByteArray) 🔗

Overwrites data of an existing **Image**. Non-static equivalent of create_from_data.

---

void **set_pixel**(x: int, y: int, color: Color)

Sets the Color of the pixel at `(x, y)` to `color`.

Example:

GDScript

```
var img_width = 10
var img_height = 5
var img = Image.create(img_width, img_height, false,
Image.FORMAT_RGBA8)

img.set_pixel(1, 2, Color.RED) # Sets the color at (1, 2) to
red.
```

C#

```
int imgWidth = 10;
int imgHeight = 5;
var img = Image.Create(imgWidth, imgHeight, false,
Image.Format.Rgba8);

img.SetPixel(1, 2, Colors.Red); // Sets the color at (1, 2) to
red.
```

This is the same as set_pixelv, but with a two integer arguments instead of a Vector2i argument.

---

void **set_pixelv**(point: Vector2i, color: Color)

Sets the Color of the pixel at `point` to `color`.

Example:

GDScript

```
var img_width = 10
var img_height = 5
var img = Image.create(img_width, img_height, false,
Image.FORMAT_RGBA8)

img.set_pixelv(Vector2i(1, 2), Color.RED) # Sets the color at
(1, 2) to red.
```

C#

```
int imgWidth = 10;
int imgHeight = 5;
var img = Image.Create(imgWidth, imgHeight, false,
Image.Format.Rgba8);

img.SetPixelv(new Vector2I(1, 2), Colors.Red); // Sets the
color at (1, 2) to red.
```

This is the same as set_pixel, but with a Vector2i argument instead of two integer arguments.

---

void **shrink_x2**() 🔗

Shrinks the image by a factor of 2 on each axis (this divides the pixel count by 4).

---

void **srgb_to_linear**() 🔗

Converts the raw data from the sRGB colorspace to a linear scale.

ImageTexture

Inherits: Texture2D **<** Texture **<** Resource **<** RefCounted **<** Object

A Texture2D based on an Image.

Description

A Texture2D based on an Image. For an image to be displayed, an **ImageTexture** has to be created from it using the create_from_image method:

```
var image = Image.load_from_file("res://icon.svg")
var texture = ImageTexture.create_from_image(image)
$Sprite2D.texture = texture
```

This way, textures can be created at run-time by loading images both from within the editor and externally.

Warning: Prefer to load imported textures with @GDScript.load over loading them from within the filesystem dynamically with Image.load, as it may not work in exported projects:

```
var texture = load("res://icon.svg")
$Sprite2D.texture = texture
```

This is because images have to be imported as a CompressedTexture2D first to be loaded with @GDScript.load. If you'd still like to load an image file just like any other Resource, import it as an Image resource instead, and then load it normally using the @GDScript.load method.

Note: The image can be retrieved from an imported texture using the Texture2D.get_image method, which returns a copy of the image:

```
var texture = load("res://icon.svg")
var image: Image = texture.get_image()
```

An **ImageTexture** is not meant to be operated from within the editor interface directly, and is mostly useful for rendering images on screen dynamically via code. If you need to generate images procedurally from within the editor, consider saving and importing images as custom texture resources implementing a new EditorImportPlugin.

Note: The maximum texture size is 16384×16384 pixels due to graphics hardware limitations.

Tutorials

- Importing images

Properties

bool	resource_local_to_scene	`false` (overrides Resource)

Methods

ImageTexture	create_from_image(image: Image) static
Format	get_format() const

void	set_image(image: Image)
void	set_size_override(size: Vector2i)
void	update(image: Image)

Method Descriptions

ImageTexture **create_from_image**(image: Image) static 🔗

Creates a new **ImageTexture** and initializes it by allocating and setting the data from an Image.

Format **get_format**() const 🔗

Returns the format of the texture, one of Format.

---

void **set_image**(image: Image) 🔗

Replaces the texture's data with a new Image. This will re-allocate new memory for the texture.

If you want to update the image, but don't need to change its parameters (format, size), use update instead for better performance.

---

void **set_size_override**(size: Vector2i) 🔗

Resizes the texture to the specified dimensions.

---

void **update**(image: Image)

Replaces the texture's data with a new Image.

Note: The texture has to be created using create_from_image or initialized first with the set_image method before it can be updated. The new image dimensions, format, and mipmaps configuration should match the existing texture's image configuration.

Use this method over set_image if you need to update the texture frequently, which is faster than allocating additional memory for a new texture each time.

ImageTexture3D

Inherits: Texture3D **<** Texture **<** Resource **<** RefCounted **<** Object

Texture with 3 dimensions.

Description

ImageTexture3D is a 3-dimensional ImageTexture that has a width, height, and depth. See also ImageTextureLayered.

3D textures are typically used to store density maps for FogMaterial, color correction LUTs for Environment, vector fields for GPUParticlesAttractorVectorField3D and collision maps for GPUParticlesCollisionSDF3D. 3D textures can also be used in custom shaders.

Methods

Error	create(format: Format, width: int, height: int, depth: int, use_mipmaps: bool, data: Array[Image])
void	update(data: Array[Image])

Method Descriptions

Error **create**(format: Format, width: int, height: int, depth: int, use_mipmaps: bool, data: Array[Image]) 🔗

Creates the **ImageTexture3D** with specified `width`, `height`, and `depth`. See Format for `format` options. If `use_mipmaps` is `true`, then generate mipmaps for the **ImageTexture3D**.

---

void **update**(data: Array[Image]) 🔗

Replaces the texture's existing data with the layers specified in `data`. The size of `data` must match the parameters that were used for create. In other words, the texture cannot be resized or have its format changed by calling update.

ImageTextureLayered

Inherits: TextureLayered **<** Texture **<** Resource **<** RefCounted **<** Object

Inherited By: Cubemap, CubemapArray, Texture2DArray

Base class for texture types which contain the data of multiple ImageTextures. Each image is of the same size and format.

Description

Base class for Texture2DArray, Cubemap and CubemapArray. Cannot be used directly, but contains all the functions necessary for accessing the derived resource types. See also Texture3D.

Methods

Error	create_from_images(images: Array[Image])
void	update_layer(image: Image, layer: int)

Method Descriptions

Error **create_from_images**(images: Array[Image]) 🔗

Creates an **ImageTextureLayered** from an array of Images. See Image.create for the expected data format. The first image decides the width, height, image format and mipmapping setting. The other images *must* have the same width, height, image format and mipmapping setting.

Each Image represents one `layer`.

---

void **update_layer**(image: Image, layer: int) 🔗

Replaces the existing Image data at the given `layer` with this new image.

The given Image must have the same width, height, image format, and mipmapping flag as the rest of the referenced images.

If the image format is unsupported, it will be decompressed and converted to a similar and supported Format.

The update is immediate: it's synchronized with drawing.

ImmediateMesh

Inherits: Mesh **<** Resource **<** RefCounted **<** Object

Mesh optimized for creating geometry manually.

Description

A mesh type optimized for creating geometry manually, similar to OpenGL 1.x immediate mode.

Here's a sample on how to generate a triangular face:

GDScript

```
var mesh = ImmediateMesh.new()
mesh.surface_begin(Mesh.PRIMITIVE_TRIANGLES)
mesh.surface_add_vertex(Vector3.LEFT)
mesh.surface_add_vertex(Vector3.FORWARD)
mesh.surface_add_vertex(Vector3.ZERO)
mesh.surface_end()
```

C#

```
var mesh = new ImmediateMesh();
mesh.SurfaceBegin(Mesh.PrimitiveType.Triangles);
mesh.SurfaceAddVertex(Vector3.Left);
mesh.SurfaceAddVertex(Vector3.Forward);
mesh.SurfaceAddVertex(Vector3.Zero);
mesh.SurfaceEnd();
```

Note: Generating complex geometries with **ImmediateMesh** is highly inefficient. Instead, it is designed to generate simple geometry that changes often.

Tutorials

- Using ImmediateMesh

Methods

void	clear_surfaces()
void	surface_add_vertex(vertex: Vector3)
void	surface_add_vertex_2d(vertex: Vector2)
void	surface_begin(primitive: PrimitiveType, material: Material = null)
void	surface_end()
void	surface_set_color(color: Color)
void	surface_set_normal(normal: Vector3)
void	surface_set_tangent(tangent: Plane)
void	surface_set_uv(uv: Vector2)
void	surface_set_uv2(uv2: Vector2)

Method Descriptions

void **clear_surfaces**()

Clear all surfaces.

---

void **surface_add_vertex**(vertex: Vector3)

Add a 3D vertex using the current attributes previously set.

---

void **surface_add_vertex_2d**(vertex: Vector2)

Add a 2D vertex using the current attributes previously set.

---

void **surface_begin**(primitive: PrimitiveType, material: Material = null)

Begin a new surface.

---

void **surface_end**()

End and commit current surface. Note that surface being created will not be visible until this function is called.

---

void **surface_set_color**(color: Color)

Set the color attribute that will be pushed with the next vertex.

---

void **surface_set_normal**(normal: Vector3)

Set the normal attribute that will be pushed with the next vertex.

---

void **surface_set_tangent**(tangent: Plane)

Set the tangent attribute that will be pushed with the next vertex.

---

void **surface_set_uv**(uv: Vector2)

Set the UV attribute that will be pushed with the next vertex.

---

void **surface_set_uv2**(uv2: Vector2)

Set the UV2 attribute that will be pushed with the next vertex.

ImporterMesh

Inherits: Resource **<** RefCounted **<** Object

A Resource that contains vertex array-based geometry during the import process.

Description

ImporterMesh is a type of Resource analogous to ArrayMesh. It contains vertex array-based geometry, divided in *surfaces*. Each surface contains a completely separate array and a material used to draw it. Design wise, a mesh with multiple surfaces is preferred to a single surface, because objects created in 3D editing software commonly contain multiple materials.

Unlike its runtime counterpart, **ImporterMesh** contains mesh data before various import steps, such as lod and shadow mesh generation, have taken place. Modify surface data by calling clear, followed by add_surface for each surface.

Properties

Dictionary	_data	`{ "surfaces": [] }`

Methods

void	add_blend_shape(name: String)

void	add_surface(primitive: PrimitiveType, arrays: Array, blend_shapes: Array[Array] = [], lods: Dictionary = {}, material: Material = null, name: String = "", flags: int = 0)
void	clear()
void	generate_lods(normal_merge_angle: float, normal_split_angle: float, bone_transform_array: Array)
int	get_blend_shape_count() const
BlendShapeMode	get_blend_shape_mode() const
String	get_blend_shape_name(blend_shape_idx: int) const
Vector2i	get_lightmap_size_hint() const
ArrayMesh	get_mesh(base_mesh: ArrayMesh = null)
Array	get_surface_arrays(surface_idx: int) const
Array	get_surface_blend_shape_arrays(surface_idx: int, blend_shape_idx: int) const

int	get_surface_count() const
int	get_surface_format(surface_idx: int) const
int	get_surface_lod_count(surface_idx: int) const
PackedInt32Array	get_surface_lod_indices(surface_idx: int, lod_idx: int) const
float	get_surface_lod_size(surface_idx: int, lod_idx: int) const
Material	get_surface_material(surface_idx: int) const
String	get_surface_name(surface_idx: int) const
PrimitiveType	get_surface_primitive_type(surface_idx: int)
void	set_blend_shape_mode(mode: BlendShapeMode)
void	set_lightmap_size_hint(size: Vector2i)
void	set_surface_material(surface_idx: int, material: Material)

void	set_surface_name(surface_idx: int, name: String)

Property Descriptions

Dictionary **_data** = `{ "surfaces": [] }` 🔗

There is currently no description for this property. Please help us by contributing one!

Method Descriptions

void **add_blend_shape**(name: String) 🔗

Adds name for a blend shape that will be added with add_surface. Must be called before surface is added.

void **add_surface**(primitive: PrimitiveType, arrays: Array, blend_shapes: Array[Array] = [], lods: Dictionary = {}, material: Material = null, name: String = "", flags: int = 0) 🔗

Creates a new surface. Mesh.get_surface_count will become the `surf_idx` for this new surface.

Surfaces are created to be rendered using a `primitive`, which may be any of the values defined in PrimitiveType.

The `arrays` argument is an array of arrays. Each of the Mesh.ARRAY_MAX elements contains an array with some of the mesh data for this surface as described by the corresponding member of ArrayType or `null` if it is not used by the surface. For example, `arrays[0]` is the array of vertices. That first vertex sub-array is always required; the others are optional. Adding an index array puts this surface into "index mode" where the vertex and other arrays become the sources of data and the index array defines the vertex order. All sub-arrays must have the same length as the vertex array (or be an exact multiple of the vertex array's length, when multiple elements of a sub-array correspond to a single vertex) or be empty, except for Mesh.ARRAY_INDEX if it is used.

The `blend_shapes` argument is an array of vertex data for each blend shape. Each element is an array of the same structure as `arrays`, but Mesh.ARRAY_VERTEX, Mesh.ARRAY_NORMAL, and Mesh.ARRAY_TANGENT are set if and only if they are set in `arrays` and all other entries are `null`.

The `lods` argument is a dictionary with float keys and PackedInt32Array values. Each entry in the dictionary represents an LOD level of the surface, where the value is the Mesh.ARRAY_INDEX array to use for the LOD level and the key is roughly proportional to the distance at which the LOD stats being used. I.e., increasing the key of an LOD also increases the distance that the objects has to be from the camera before the LOD is used.

The `flags` argument is the bitwise or of, as required: One value of ArrayCustomFormat left shifted by `ARRAY_FORMAT_CUSTOMn_SHIFT` for each custom channel in use, Mesh.ARRAY_FLAG_USE_DYNAMIC_UPDATE, Mesh.ARRAY_FLAG_USE_8_BONE_WEIGHTS, or Mesh.ARRAY_FLAG_USES_EMPTY_VERTEX_ARRAY.

Note: When using indices, it is recommended to only use points, lines, or triangles.

---

void **clear**() 🔗

Removes all surfaces and blend shapes from this **ImporterMesh**.

---

void **generate_lods**(normal_merge_angle: float, normal_split_angle: float, bone_transform_array: Array)

Generates all lods for this ImporterMesh.

`normal_merge_angle` and `normal_split_angle` are in degrees and used in the same way as the importer settings in `lods`. As a good default, use 25 and 60 respectively.

The number of generated lods can be accessed using get_surface_lod_count, and each LOD is available in get_surface_lod_size and get_surface_lod_indices.

`bone_transform_array` is an Array which can be either empty or contain Transform3Ds which, for each of the mesh's bone IDs, will apply mesh skinning when generating the LOD mesh variations. This is usually used to account for discrepancies in scale between the mesh itself and its skinning data.

---

int **get_blend_shape_count**() const

Returns the number of blend shapes that the mesh holds.

BlendShapeMode **get_blend_shape_mode**() const

Returns the blend shape mode for this Mesh.

String **get_blend_shape_name**(blend_shape_idx: int) const

Returns the name of the blend shape at this index.

Vector2i **get_lightmap_size_hint**() const

Returns the size hint of this mesh for lightmap-unwrapping in UV-space.

ArrayMesh **get_mesh**(base_mesh: ArrayMesh = null)

Returns the mesh data represented by this **ImporterMesh** as a usable ArrayMesh.

This method caches the returned mesh, and subsequent calls will return the cached data until clear is called.

If not yet cached and `base_mesh` is provided, `base_mesh` will be used and mutated.

Array **get_surface_arrays**(surface_idx: int) const

Returns the arrays for the vertices, normals, UVs, etc. that make up the requested surface. See add_surface.

Array **get_surface_blend_shape_arrays**(surface_idx: int, blend_shape_idx: int) const

Returns a single set of blend shape arrays for the requested blend shape index for a surface.

---

int **get_surface_count**() const

Returns the number of surfaces that the mesh holds.

---

int **get_surface_format**(surface_idx: int) const

Returns the format of the surface that the mesh holds.

---

int **get_surface_lod_count**(surface_idx: int) const 🔗

Returns the number of lods that the mesh holds on a given surface.

PackedInt32Array **get_surface_lod_indices**(surface_idx: int, lod_idx: int) const 🔗

Returns the index buffer of a lod for a surface.

---

float **get_surface_lod_size**(surface_idx: int, lod_idx: int) const 🔗

Returns the screen ratio which activates a lod for a surface.

Material **get_surface_material**(surface_idx: int) const 🔗

Returns a Material in a given surface. Surface is rendered using this material.

String **get_surface_name**(surface_idx: int) const 🔗

Gets the name assigned to this surface.

PrimitiveType **get_surface_primitive_type**(surface_idx: int) 🔗

Returns the primitive type of the requested surface (see add_surface).

---

void **set_blend_shape_mode**(mode: BlendShapeMode) 🔗

Sets the blend shape mode to one of BlendShapeMode.

---

void **set_lightmap_size_hint**(size: Vector2i) 🔗

Sets the size hint of this mesh for lightmap-unwrapping in UV-space.

---

void **set_surface_material**(surface_idx: int, material: Material) 🔗

Sets a Material for a given surface. Surface will be rendered using this material.

---

void **set_surface_name**(surface_idx: int, name: String) 🔗

Sets a name for a given surface.

InputEvent

Inherits: Resource **<** RefCounted **<** Object

Inherited By: InputEventAction, InputEventFromWindow, InputEventJoypadButton, InputEventJoypadMotion, InputEventMIDI, InputEventShortcut

Abstract base class for input events.

Description

Abstract base class of all types of input events. See Node._input.

Tutorials

- Using InputEvent
- Viewport and canvas transforms
- 2D Dodge The Creeps Demo [https://godotengine.org/asset-library/asset/2712]
- 3D Voxel Demo [https://godotengine.org/asset-library/asset/2755]

Properties

int	device	`0`

Methods

bool	accumulate(with_event: InputEvent)
String	as_text() const
float	get_action_strength(action: StringName, exact_match: bool = false) const
bool	is_action(action: StringName, exact_match: bool = false) const
bool	is_action_pressed(action: StringName, allow_echo: bool = false, exact_match: bool = false) const
bool	is_action_released(action: StringName, exact_match: bool = false) const
bool	is_action_type() const
bool	is_canceled() const
bool	is_echo() const
bool	is_match(event: InputEvent, exact_match: bool = true) const
bool	is_pressed() const

bool	is_released() const
InputEvent	xformed_by(xform: Transform2D, local_ofs: Vector2 = Vector2(0, 0)) const

Constants

DEVICE_ID_EMULATION = `-1`

Device ID used for emulated mouse input from a touchscreen, or for emulated touch input from a mouse. This can be used to distinguish emulated mouse input from physical mouse input, or emulated touch input from physical touch input.

Property Descriptions

int **device** = `0`

- void **set_device**(value: int)
- int **get_device**()

The event's device ID.

Note: device can be negative for special use cases that don't refer to devices physically present on the system. See DEVICE_ID_EMULATION.

Method Descriptions

bool **accumulate**(with_event: InputEvent) 🔗

Returns `true` if the given input event and this input event can be added together (only for events of type InputEventMouseMotion).

The given input event's position, global position and speed will be copied. The resulting `relative` is a sum of both events. Both events' modifiers have to be identical.

String **as_text**() const 🔗

Returns a String representation of the event.

---

float **get_action_strength**(action: StringName, exact_match: bool = false) const 🔗

Returns a value between 0.0 and 1.0 depending on the given actions' state. Useful for getting the value of events of type InputEventJoypadMotion.

If `exact_match` is `false`, it ignores additional input modifiers for InputEventKey and InputEventMouseButton events, and the direction for InputEventJoypadMotion events.

---

bool **is_action**(action: StringName, exact_match: bool = false) const 🔗

Returns `true` if this input event matches a pre-defined action of any type.

If `exact_match` is `false`, it ignores additional input modifiers for InputEventKey and InputEventMouseButton events, and the direction for InputEventJoypadMotion events.

---

bool **is_action_pressed**(action: StringName, allow_echo: bool = false, exact_match: bool = false) const 🔗

Returns `true` if the given action is being pressed (and is not an echo event for InputEventKey events, unless `allow_echo` is `true`). Not relevant for events of type InputEventMouseMotion or InputEventScreenDrag.

If `exact_match` is `false`, it ignores additional input modifiers for InputEventKey and InputEventMouseButton events, and the direction for InputEventJoypadMotion events.

Note: Due to keyboard ghosting, is_action_pressed may return `false` even if one of the action's keys is pressed. See Input examples in the documentation for more information.

---

bool **is_action_released**(action: StringName, exact_match: bool = false) const 🔗

Returns `true` if the given action is released (i.e. not pressed). Not relevant for events of type InputEventMouseMotion or InputEventScreenDrag.

If `exact_match` is `false`, it ignores additional input modifiers for InputEventKey and InputEventMouseButton events, and the direction for InputEventJoypadMotion events.

---

bool **is_action_type**() const 🔗

Returns `true` if this input event's type is one that can be assigned to an input action.

---

bool **is_canceled**() const

Returns `true` if this input event has been canceled.

---

bool **is_echo**() const

Returns `true` if this input event is an echo event (only for events of type InputEventKey). An echo event is a repeated key event sent when the user is holding down the key. Any other event type returns `false`.

Note: The rate at which echo events are sent is typically around 20 events per second (after holding down the key for roughly half a second). However, the key repeat delay/speed can be changed by the user or disabled entirely in the operating system settings. To ensure your project works correctly on all configurations, do not assume the user has a specific key repeat configuration in your project's behavior.

---

bool **is_match**(event: InputEvent, exact_match: bool = true) const

Returns `true` if the specified `event` matches this event. Only valid for action events i.e key (InputEventKey), button (InputEventMouseButton or InputEventJoypadButton), axis InputEventJoypadMotion or action (InputEventAction) events.

If `exact_match` is `false`, it ignores additional input modifiers for InputEventKey and InputEventMouseButton events, and the direction for InputEventJoypadMotion events.

---

bool **is_pressed**() const

Returns `true` if this input event is pressed. Not relevant for events of type InputEventMouseMotion or InputEventScreenDrag.

Note: Due to keyboard ghosting, is_pressed may return `false` even if one of the action's keys is pressed. See Input examples in the documentation for more information.

---

bool **is_released**() const 🔗

Returns `true` if this input event is released. Not relevant for events of type InputEventMouseMotion or InputEventScreenDrag.

InputEvent **xformed_by**(xform: Transform2D, local_ofs: Vector2 = Vector2(0, 0)) const 🔗

Returns a copy of the given input event which has been offset by `local_ofs` and transformed by `xform`. Relevant for events of type InputEventMouseButton, InputEventMouseMotion, InputEventScreenTouch, InputEventScreenDrag, InputEventMagnifyGesture and InputEventPanGesture.

InputEventAction

Inherits: InputEvent **<** Resource **<** RefCounted **<** Object

An input event type for actions.

Description

Contains a generic action which can be targeted from several types of inputs. Actions and their events can be set in the **Input Map** tab in **Project > Project Settings**, or with the InputMap class.

Note: Unlike the other InputEvent subclasses which map to unique physical events, this virtual one is not emitted by the engine. This class is useful to emit actions manually with Input.parse_input_event, which are then received in Node._input. To check if a physical event matches an action from the Input Map, use InputEvent.is_action and InputEvent.is_action_pressed.

Tutorials

- Using InputEvent: Actions
- 2D Dodge The Creeps Demo [https://godotengine.org/asset-library/asset/2712]
- 3D Voxel Demo [https://godotengine.org/asset-library/asset/2755]

Properties

StringName	action	`&""`

int	event_index	`-1`
bool	pressed	`false`
float	strength	`1.0`

Property Descriptions

StringName **action** = `&""` 🔗

- void **set_action**(value: StringName)
- StringName **get_action**()

The action's name. Actions are accessed via this String.

---

int **event_index** = `-1` 🔗

- void **set_event_index**(value: int)
- int **get_event_index**()

The real event index in action this event corresponds to (from events defined for this action in the InputMap). If `-1`, a unique ID will be used and actions pressed with this ID will need to be released with another **InputEventAction**.

---

bool **pressed** = `false` 🔗

- void **set_pressed**(value: bool)
- bool **is_pressed**()

If `true`, the action's state is pressed. If `false`, the action's state is released.

---

float **strength** = `1.0`

- void **set_strength**(value: float)
- float **get_strength**()

The action's strength between 0 and 1. This value is considered as equal to 0 if pressed is `false`. The event strength allows faking analog joypad motion events, by specifying how strongly the joypad axis is bent or pressed.

InputEventFromWindow

Inherits: InputEvent **<** Resource **<** RefCounted **<** Object

Inherited By: InputEventScreenDrag, InputEventScreenTouch, InputEventWithModifiers

Abstract base class for Viewport-based input events.

Description

InputEventFromWindow represents events specifically received by windows. This includes mouse events, keyboard events in focused windows or touch screen actions.

Properties

int	window_id	`0`

Property Descriptions

int **window_id** = `0` 🔗

- void **set_window_id**(value: int)
- int **get_window_id**()

The ID of a Window that received this event.

InputEventGesture

Inherits: InputEventWithModifiers **<** InputEventFromWindow **<** InputEvent **<** Resource **<** RefCounted **<** Object

Inherited By: InputEventMagnifyGesture, InputEventPanGesture

Abstract base class for touch gestures.

Description

InputEventGestures are sent when a user performs a supported gesture on a touch screen. Gestures can't be emulated using mouse, because they typically require multi-touch.

Tutorials

- Using InputEvent

Properties

Vector2	position	`Vector2(0, 0)`

Property Descriptions

Vector2 **position** = `Vector2(0, 0)` 🔗

- void **set_position**(value: Vector2)
- Vector2 **get_position**()

The local gesture position relative to the Viewport. If used in Control._gui_input, the position is relative to the current Control that received this gesture.

InputEventJoypadButton

Inherits: InputEvent **<** Resource **<** RefCounted **<** Object

Represents a gamepad button being pressed or released.

Description

Input event type for gamepad buttons. For gamepad analog sticks and joysticks, see InputEventJoypadMotion.

Tutorials

- Using InputEvent

Properties

JoyButton	button_index	`0`
bool	pressed	`false`
float	pressure	`0.0`

Property Descriptions

JoyButton **button_index** = `0`

- void **set_button_index**(value: JoyButton)
- JoyButton **get_button_index**()

Button identifier. One of the JoyButton button constants.

---

bool **pressed** = `false`

- void **set_pressed**(value: bool)
- bool **is_pressed**()

If `true`, the button's state is pressed. If `false`, the button's state is released.

---

float **pressure** = `0.0`

- void **set_pressure**(value: float)
- float **get_pressure**()

Deprecated: This property is never set by the engine and is always `0`.

InputEventJoypadMotion

Inherits: InputEvent **<** Resource **<** RefCounted **<** Object

Represents axis motions (such as joystick or analog triggers) from a gamepad.

Description

Stores information about joystick motions. One **InputEventJoypadMotion** represents one axis at a time. For gamepad buttons, see InputEventJoypadButton.

Tutorials

- Using InputEvent

Properties

JoyAxis	axis	`0`
float	axis_value	`0.0`

Property Descriptions

JoyAxis **axis** = `0` 🔗

- void **set_axis**(value: JoyAxis)
- JoyAxis **get_axis**()

Axis identifier. Use one of the JoyAxis axis constants.

---

float **axis_value** = `0.0` 🔗

- void **set_axis_value**(value: float)
- float **get_axis_value**()

Current position of the joystick on the given axis. The value ranges from `-1.0` to `1.0`. A value of `0` means the axis is in its resting position.

InputEventKey

Inherits: InputEventWithModifiers **<** InputEventFromWindow **<** InputEvent **<** Resource **<** RefCounted **<** Object

Represents a key on a keyboard being pressed or released.

Description

An input event for keys on a keyboard. Supports key presses, key releases and echo events. It can also be received in Node._unhandled_key_input.

Note: Events received from the keyboard usually have all properties set. Event mappings should have only one of the keycode, physical_keycode or unicode set.

When events are compared, properties are checked in the following priority - keycode, physical_keycode and unicode. Events with the first matching value will be considered equal.

Tutorials

- Using InputEvent

Properties

bool	echo	`false`

Key	key_label	`0`
Key	keycode	`0`
KeyLocation	location	`0`
Key	physical_keycode	`0`
bool	pressed	`false`
int	unicode	`0`

Methods

String	as_text_key_label() const
String	as_text_keycode() const
String	as_text_location() const
String	as_text_physical_keycode() const
Key	get_key_label_with_modifiers() const

Key	get_keycode_with_modifiers() const
Key	get_physical_keycode_with_modifiers() const

Property Descriptions

bool **echo** = `false` 🔗

- void **set_echo**(value: bool)
- bool **is_echo**()

If `true`, the key was already pressed before this event. An echo event is a repeated key event sent when the user is holding down the key.

Note: The rate at which echo events are sent is typically around 20 events per second (after holding down the key for roughly half a second). However, the key repeat delay/speed can be changed by the user or disabled entirely in the operating system settings. To ensure your project works correctly on all configurations, do not assume the user has a specific key repeat configuration in your project's behavior.

Key **key_label** = `0` 🔗

- void **set_key_label**(value: Key)
- Key **get_key_label**()

Represents the localized label printed on the key in the current keyboard layout, which corresponds to one of the Key constants or any valid Unicode character.

For keyboard layouts with a single label on the key, it is equivalent to keycode.

To get a human-readable representation of the **InputEventKey**, use `OS.get_keycode_string(event.key_label)` where `event` is the **InputEventKey**.

```
+------+ +------+
| Q    | | Q    | - "Q" - keycode
|    Й | |    ض | - "Й" and "ض" - key_label
+------+ +------+
```

Key **keycode** = `0`

- void **set_keycode**(value: Key)
- Key **get_keycode**()

Latin label printed on the key in the current keyboard layout, which corresponds to one of the Key constants.

To get a human-readable representation of the **InputEventKey**, use `OS.get_keycode_string(event.keycode)` where `event` is the **InputEventKey**.

```
+------+ +------+
| Q    | | Q    | - "Q" - keycode
|    Й | |    ض | - "Й" and "ض" - key_label
+------+ +------+
```

KeyLocation **location** = `0`

- void **set_location**(value: KeyLocation)
- KeyLocation **get_location**()

Represents the location of a key which has both left and right versions, such as `Shift` or `Alt`.

Key **physical_keycode** = `0`

- void **set_physical_keycode**(value: Key)
- Key **get_physical_keycode**()

Represents the physical location of a key on the 101/102-key US QWERTY keyboard, which corresponds to one of the Key constants.

To get a human-readable representation of the **InputEventKey**, use OS.get_keycode_string in combination with DisplayServer.keyboard_get_keycode_from_physical:

GDScript

```
func _input(event):
    if event is InputEventKey:
        var keycode = DisplayServer.keyboard_get_keycode_from_physical(event.physical_keycode)
        print(OS.get_keycode_string(keycode))
```

C#

```
public override void _Input(InputEvent @event)
{
    if (@event is InputEventKey inputEventKey)
    {
        var keycode = DisplayServer.KeyboardGetKeycodeFromPhysical(inputEventKey.PhysicalKeycode);
        GD.Print(OS.GetKeycodeString(keycode));
    }
}
```

---

bool **pressed** = `false`

- void **set_pressed**(value: bool)
- bool **is_pressed**()

If `true`, the key's state is pressed. If `false`, the key's state is released.

---

int **unicode** = `0` 🔗

- void **set_unicode**(value: int)
- int **get_unicode**()

The key Unicode character code (when relevant), shifted by modifier keys. Unicode character codes for composite characters and complex scripts may not be available unless IME input mode is active. See Window.set_ime_active for more information.

Method Descriptions

String **as_text_key_label**() const 🔗

Returns a String representation of the event's key_label and modifiers.

String **as_text_keycode**() const 🔗

Returns a String representation of the event's keycode and modifiers.

String **as_text_location**() const 🔗

Returns a String representation of the event's location. This will be a blank string if the event is not specific to a location.

String **as_text_physical_keycode**() const 🔗

Returns a String representation of the event's physical_keycode and modifiers.

Key **get_key_label_with_modifiers**() const 🔗

Returns the localized key label combined with modifier keys such as `Shift` or `Alt`. See also InputEventWithModifiers.

To get a human-readable representation of the **InputEventKey** with modifiers, use `OS.get_keycode_string(event.get_key_label_with_modifiers())` where `event` is the **InputEventKey**.

Key **get_keycode_with_modifiers**() const 🔗

Returns the Latin keycode combined with modifier keys such as `Shift` or `Alt`. See also InputEventWithModifiers.

To get a human-readable representation of the **InputEventKey** with modifiers, use `OS.get_keycode_string(event.get_keycode_with_modifiers())` where `event` is the **InputEventKey**.

Key **get_physical_keycode_with_modifiers**() const 🔗

Returns the physical keycode combined with modifier keys such as `Shift` or `Alt`. See also InputEventWithModifiers.

To get a human-readable representation of the **InputEventKey** with modifiers, use `OS.get_keycode_string(event.get_physical_keycode_with_modifiers())` where `event` is the **InputEventKey**.

InputEventMagnifyGesture

Inherits: InputEventGesture **<** InputEventWithModifiers **<** InputEventFromWindow **<** InputEvent **<** Resource **<** RefCounted **<** Object

Represents a magnifying touch gesture.

Description

Stores the factor of a magnifying touch gesture. This is usually performed when the user pinches the touch screen and used for zooming in/out.

Note: On Android, this requires the ProjectSettings.input_devices/pointing/android/enable_pan_and_scale_gestures project setting to be enabled.

Tutorials

- Using InputEvent

Properties

float	factor	`1.0`

Property Descriptions

float **factor** = `1.0` 🔗

- void **set_factor**(value: float)
- float **get_factor**()

The amount (or delta) of the event. This value is closer to `1.0` the slower the gesture is performed.

InputEventMIDI

Inherits: InputEvent **<** Resource **<** RefCounted **<** Object

Represents a MIDI message from a MIDI device, such as a musical keyboard.

Description

InputEventMIDI stores information about messages from MIDI [https://en.wikipedia.org/wiki/MIDI] (Musical Instrument Digital Interface) devices. These may include musical keyboards, synthesizers, and drum machines.

MIDI messages can be received over a 5-pin MIDI connector or over USB. If your device supports both be sure to check the settings in the device to see which output it is using.

By default, Godot does not detect MIDI devices. You need to call OS.open_midi_inputs, first. You can check which devices are detected with OS.get_connected_midi_inputs, and close the connection with OS.close_midi_inputs.

GDScript

```
func _ready():
    OS.open_midi_inputs()
    print(OS.get_connected_midi_inputs())

func _input(input_event):
    if input_event is InputEventMIDI:
        _print_midi_info(input_event)

func _print_midi_info(midi_event):
    print(midi_event)
    print("Channel ", midi_event.channel)
    print("Message ", midi_event.message)
    print("Pitch ", midi_event.pitch)
    print("Velocity ", midi_event.velocity)
    print("Instrument ", midi_event.instrument)
    print("Pressure ", midi_event.pressure)
    print("Controller number: ", midi_event.controller_number)
    print("Controller value: ", midi_event.controller_value)
```

C#

```
public override void _Ready()
{
    OS.OpenMidiInputs();
    GD.Print(OS.GetConnectedMidiInputs());
}

public override void _Input(InputEvent inputEvent)
{
    if (inputEvent is InputEventMidi midiEvent)
    {
        PrintMIDIInfo(midiEvent);
    }
}

private void PrintMIDIInfo(InputEventMidi midiEvent)
{
    GD.Print(midiEvent);
    GD.Print($"Channel {midiEvent.Channel}");
    GD.Print($"Message {midiEvent.Message}");
    GD.Print($"Pitch {midiEvent.Pitch}");
    GD.Print($"Velocity {midiEvent.Velocity}");
    GD.Print($"Instrument {midiEvent.Instrument}");
    GD.Print($"Pressure {midiEvent.Pressure}");
    GD.Print($"Controller number:
{midiEvent.ControllerNumber}");
    GD.Print($"Controller value: {midiEvent.ControllerValue}");
}
```

Note: Godot does not support MIDI output, so there is no way to emit MIDI messages from Godot. Only MIDI input is supported.

Tutorials

- MIDI Message Status Byte List [https://www.midi.org/specifications-old/item/table-2-expanded-messages-list-status-bytes]
- Wikipedia General MIDI Instrument List [https://en.wikipedia.org/wiki/General_MIDI#Program_change_events]
- Wikipedia Piano Key Frequencies List [https://en.wikipedia.org/wiki/Piano_key_frequencies#List]

Properties

int	channel	0
int	controller_number	0
int	controller_value	0
int	instrument	0
MIDIMessage	message	0
int	pitch	0
int	pressure	0
int	velocity	0

Property Descriptions

int **channel** = `0` 🔗

- void **set_channel**(value: int)
- int **get_channel**()

The MIDI channel of this message, ranging from `0` to `15`. MIDI channel `9` is reserved for percussion instruments.

---

int **controller_number** = `0` 🔗

- void **set_controller_number**(value: int)
- int **get_controller_number**()

The unique number of the controller, if message is @GlobalScope.MIDI_MESSAGE_CONTROL_CHANGE, otherwise this is `0`. This value can be used to identify sliders for volume, balance, and panning, as well as switches and pedals on the MIDI device. See the General MIDI specification [https://en.wikipedia.org/wiki/General_MIDI#Controller_events] for a small list.

---

int **controller_value** = `0` 🔗

- void **set_controller_value**(value: int)
- int **get_controller_value**()

The value applied to the controller. If message is @GlobalScope.MIDI_MESSAGE_CONTROL_CHANGE, this value ranges from `0` to `127`, otherwise it is `0`. See also controller_value.

---

int **instrument** = `0` 🔗

- void **set_instrument**(value: int)
- int **get_instrument**()

The instrument (also called *program* or *preset*) used on this MIDI message. This value ranges from `0` to `127`.

To see what each value means, refer to the General MIDI's instrument list [https://en.wikipedia.org/wiki/General_MIDI#Program_change_events]. Keep in mind that the list is off by 1 because it does not begin from 0. A value of `0` corresponds to the acoustic grand piano.

MIDIMessage **message** = `0` 🔗

- void **set_message**(value: MIDIMessage)
- MIDIMessage **get_message**()

Represents the type of MIDI message (see the MIDIMessage enum).

For more information, see the MIDI message status byte list chart [https://www.midi.org/specifications-old/item/table-2-expanded-messages-list-status-bytes].

int **pitch** = `0` 🔗

- void **set_pitch**(value: int)
- int **get_pitch**()

The pitch index number of this MIDI message. This value ranges from `0` to `127`.

On a piano, the **middle C** is `60`, followed by a **C-sharp** (`61`), then a **D** (`62`), and so on. Each octave is split in offsets of 12. See the "MIDI

InputEventMouse

Inherits: InputEventWithModifiers **<** InputEventFromWindow **<** InputEvent **<** Resource **<** RefCounted **<** Object

Inherited By: InputEventMouseButton, InputEventMouseMotion

Base input event type for mouse events.

Description

Stores general information about mouse events.

Tutorials

- Using InputEvent

Properties

BitField[MouseButtonMask]	button_mask	`0`
Vector2	global_position	`Vector2(0, 0)`
Vector2	position	`Vector2(0, 0)`

Property Descriptions

BitField[MouseButtonMask] **button_mask** = `0` 🔗

- void **set_button_mask**(value: BitField[MouseButtonMask])
- BitField[MouseButtonMask] **get_button_mask**()

The mouse button mask identifier, one of or a bitwise combination of the MouseButton button masks.

Vector2 **global_position** = `Vector2(0, 0)` 🔗

- void **set_global_position**(value: Vector2)
- Vector2 **get_global_position**()

When received in Node._input or Node._unhandled_input, returns the mouse's position in the root Viewport using the coordinate system of the root Viewport.

When received in Control._gui_input, returns the mouse's position in the CanvasLayer that the Control is in using the coordinate system of the CanvasLayer.

Vector2 **position** = `Vector2(0, 0)` 🔗

- void **set_position**(value: Vector2)
- Vector2 **get_position**()

When received in Node._input or Node._unhandled_input, returns the mouse's position in the Viewport this Node is in using the coordinate system of this Viewport.

When received in Control._gui_input, returns the mouse's position in the Control using the local coordinate system of the Control.

InputEventMouseButton

Inherits: InputEventMouse **<** InputEventWithModifiers **<** InputEventFromWindow **<** InputEvent **<** Resource **<** RefCounted **<** Object

Represents a mouse button being pressed or released.

Description

Stores information about mouse click events. See Node._input.

Note: On Wear OS devices, rotary input is mapped to @GlobalScope.MOUSE_BUTTON_WHEEL_UP and @GlobalScope.MOUSE_BUTTON_WHEEL_DOWN. This can be changed to @GlobalScope.MOUSE_BUTTON_WHEEL_LEFT and @GlobalScope.MOUSE_BUTTON_WHEEL_RIGHT with the ProjectSettings.input_devices/pointing/android/rotary_input_scroll_axis setting.

Tutorials

- Using InputEvent
- Mouse and input coordinates

Properties

MouseButton	button_index	0

bool	canceled	`false`
bool	double_click	`false`
float	factor	`1.0`
bool	pressed	`false`

Property Descriptions

MouseButton **button_index** = `0`

- void **set_button_index**(value: MouseButton)
- MouseButton **get_button_index**()

The mouse button identifier, one of the MouseButton button or button wheel constants.

---

bool **canceled** = `false`

- void **set_canceled**(value: bool)
- bool **is_canceled**()

If `true`, the mouse button event has been canceled.

---

bool **double_click** = `false`

- void **set_double_click**(value: bool)

- bool **is_double_click**()

If `true`, the mouse button's state is a double-click.

---

bool **factor** = `1.0`

- void **set_factor**(value: float)
- float **get_factor**()

The amount (or delta) of the event. When used for high-precision scroll events, this indicates the scroll amount (vertical or horizontal). This is only supported on some platforms; the reported sensitivity varies depending on the platform. May be `0` if not supported.

---

bool **pressed** = `false`

- void **set_pressed**(value: bool)
- bool **is_pressed**()

If `true`, the mouse button's state is pressed. If `false`, the mouse button's state is released.

InputEventMouseMotion

Inherits: InputEventMouse **<** InputEventWithModifiers **<** InputEventFromWindow **<** InputEvent **<** Resource **<** RefCounted **<** Object

Represents a mouse or a pen movement.

Description

Stores information about a mouse or a pen motion. This includes relative position, absolute position, and velocity. See Node._input.

Note: By default, this event is only emitted once per frame rendered at most. If you need more precise input reporting, set Input.use_accumulated_input to `false` to make events emitted as often as possible. If you use InputEventMouseMotion to draw lines, consider implementing Bresenham's line algorithm [https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm] as well to avoid visible gaps in lines if the user is moving the mouse quickly.

Tutorials

- Using InputEvent
- Mouse and input coordinates
- 3D Voxel Demo [https://godotengine.org/asset-library/asset/2755]

Properties

bool	pen_inverted	`false`

float	pressure	`0.0`
Vector2	relative	`Vector2(0, 0)`
Vector2	screen_relative	`Vector2(0, 0)`
Vector2	screen_velocity	`Vector2(0, 0)`
Vector2	tilt	`Vector2(0, 0)`
Vector2	velocity	`Vector2(0, 0)`

Property Descriptions

bool **pen_inverted** = `false`

- void **set_pen_inverted**(value: bool)
- bool **get_pen_inverted**()

Returns `true` when using the eraser end of a stylus pen.

Note: This property is implemented on Linux, macOS and Windows.

---

float **pressure** = `0.0`

- void **set_pressure**(value: float)
- float **get_pressure**()

Represents the pressure the user puts on the pen. Ranges from `0.0` to `1.0`.

Vector2 **relative** = `Vector2(0, 0)` 🔗

- void **set_relative**(value: Vector2)
- Vector2 **get_relative**()

The mouse position relative to the previous position (position at the last frame).

Note: Since **InputEventMouseMotion** is only emitted when the mouse moves, the last event won't have a relative position of `Vector2(0, 0)` when the user stops moving the mouse.

Note: relative is automatically scaled according to the content scale factor, which is defined by the project's stretch mode settings. This means mouse sensitivity will appear different depending on resolution when using relative in a script that handles mouse aiming with the Input.MOUSE_MODE_CAPTURED mouse mode. To avoid this, use screen_relative instead.

Vector2 **screen_relative** = `Vector2(0, 0)` 🔗

- void **set_screen_relative**(value: Vector2)
- Vector2 **get_screen_relative**()

The unscaled mouse position relative to the previous position in the coordinate system of the screen (position at the last frame).

Note: Since **InputEventMouseMotion** is only emitted when the mouse moves, the last event won't have a relative position of `Vector2(0, 0)` when the user stops moving the mouse. This coordinate is *not* scaled according to the content scale factor or calls to InputEvent.xformed_by. This should be preferred over relative for mouse aiming when using the Input.MOUSE_MODE_CAPTURED mouse mode, regardless of the project's stretch mode.

Vector2 **screen_velocity** = `Vector2(0, 0)`

- void **set_screen_velocity**(value: Vector2)
- Vector2 **get_screen_velocity**()

The unscaled mouse velocity in pixels per second in screen coordinates. This velocity is *not* scaled according to the content scale factor or calls to InputEvent.xformed_by. This should be preferred over velocity for mouse aiming when using the Input.MOUSE_MODE_CAPTURED mouse mode, regardless of the project's stretch mode.

Vector2 **tilt** = `Vector2(0, 0)`

- void **set_tilt**(value: Vector2)
- Vector2 **get_tilt**()

Represents the angles of tilt of the pen. Positive X-coordinate value indicates a tilt to the right. Positive Y-coordinate value indicates a tilt toward the user. Ranges from `-1.0` to `1.0` for both axes.

Vector2 **velocity** = `Vector2(0, 0)`

- void **set_velocity**(value: Vector2)
- Vector2 **get_velocity**()

The mouse velocity in pixels per second.

Note: velocity is automatically scaled according to the content scale factor, which is defined by the project's stretch mode settings. This means mouse sensitivity will appear different depending on resolution when using velocity in a script that handles mouse aiming with the Input.MOUSE_MODE_CAPTURED mouse mode. To avoid this, use screen_velocity instead.

InputEventPanGesture

Inherits: InputEventGesture **<** InputEventWithModifiers **<** InputEventFromWindow **<** InputEvent **<** Resource **<** RefCounted **<** Object

Represents a panning touch gesture.

Description

Stores information about pan gestures. A pan gesture is performed when the user swipes the touch screen with two fingers. It's typically used for panning/scrolling.

Note: On Android, this requires the ProjectSettings.input_devices/pointing/android/enable_pan_and_scale_gestures project setting to be enabled.

Tutorials

- Using InputEvent

Properties

Vector2	delta	`Vector2(0, 0)`

Property Descriptions

Vector2 **delta** = `Vector2(0, 0)`

- void **set_delta**(value: Vector2)
- Vector2 **get_delta**()

Panning amount since last pan event.

InputEventScreenDrag

Inherits: InputEventFromWindow **<** InputEvent **<** Resource **<** RefCounted **<** Object

Represents a screen drag event.

Description

Stores information about screen drag events. See Node._input.

Tutorials

- Using InputEvent

Properties

int	index	`0`
bool	pen_inverted	`false`
Vector2	position	`Vector2(0, 0)`
float	pressure	`0.0`
Vector2	relative	`Vector2(0, 0)`

Vector2	screen_relative	`Vector2(0, 0)`
Vector2	screen_velocity	`Vector2(0, 0)`
Vector2	tilt	`Vector2(0, 0)`
Vector2	velocity	`Vector2(0, 0)`

Property Descriptions

int **index** = `0`

- void **set_index**(value: int)
- int **get_index**()

The drag event index in the case of a multi-drag event.

---

bool **pen_inverted** = `false`

- void **set_pen_inverted**(value: bool)
- bool **get_pen_inverted**()

Returns `true` when using the eraser end of a stylus pen.

Vector2 **position** = `Vector2(0, 0)`

- void **set_position**(value: Vector2)
- Vector2 **get_position**()

The drag position in the viewport the node is in, using the coordinate system of this viewport.

---

float **pressure** = `0.0` 🔗

- void **set_pressure**(value: float)
- float **get_pressure**()

Represents the pressure the user puts on the pen. Ranges from `0.0` to `1.0`.

Vector2 **relative** = `Vector2(0, 0)` 🔗

- void **set_relative**(value: Vector2)
- Vector2 **get_relative**()

The drag position relative to the previous position (position at the last frame).

Note: relative is automatically scaled according to the content scale factor, which is defined by the project's stretch mode settings. This means touch sensitivity will appear different depending on resolution when using relative in a script that handles touch aiming. To avoid this, use screen_relative instead.

Vector2 **screen_relative** = `Vector2(0, 0)` 🔗

- void **set_screen_relative**(value: Vector2)
- Vector2 **get_screen_relative**()

The unscaled drag position relative to the previous position in screen coordinates (position at the last frame). This position is *not* scaled according to the content scale factor or calls to

InputEvent.xformed_by. This should be preferred over relative for touch aiming regardless of the project's stretch mode.

Vector2 **screen_velocity** = `Vector2(0, 0)` 🔗

- void **set_screen_velocity**(value: Vector2)
- Vector2 **get_screen_velocity**()

The unscaled drag velocity in pixels per second in screen coordinates. This velocity is *not* scaled according to the content scale factor or calls to InputEvent.xformed_by. This should be preferred over velocity for touch aiming regardless of the project's stretch mode.

Vector2 **tilt** = `Vector2(0, 0)` 🔗

- void **set_tilt**(value: Vector2)
- Vector2 **get_tilt**()

Represents the angles of tilt of the pen. Positive X-coordinate value indicates a tilt to the right. Positive Y-coordinate value indicates a tilt toward the user. Ranges from `-1.0` to `1.0` for both axes.

Vector2 **velocity** = `Vector2(0, 0)` 🔗

- void **set_velocity**(value: Vector2)
- Vector2 **get_velocity**()

The drag velocity.

Note: velocity is automatically scaled according to the content scale factor, which is defined by the project's stretch mode settings. This means touch sensitivity will appear different depending on resolution when using velocity in a script that handles touch aiming. To avoid this, use screen_velocity instead.

InputEventScreenTouch

Inherits: InputEventFromWindow **<** InputEvent **<** Resource **<** RefCounted **<** Object

Represents a screen touch event.

Description

Stores information about multi-touch press/release input events. Supports touch press, touch release and index for multi-touch count and order.

Tutorials

- Using InputEvent

Properties

bool	canceled	`false`
bool	double_tap	`false`
int	index	`0`
Vector2	position	`Vector2(0, 0)`

bool	pressed	`false`

Property Descriptions

bool **canceled** = `false`

- void **set_canceled**(value: bool)
- bool **is_canceled**()

If `true`, the touch event has been canceled.

bool **double_tap** = `false`

- void **set_double_tap**(value: bool)
- bool **is_double_tap**()

If `true`, the touch's state is a double tap.

int **index** = `0`

- void **set_index**(value: int)
- int **get_index**()

The touch index in the case of a multi-touch event. One index = one finger.

Vector2 **position** = `Vector2(0, 0)`

- void **set_position**(value: Vector2)
- Vector2 **get_position**()

The touch position in the viewport the node is in, using the coordinate system of this viewport.

---

bool **pressed** = `false` 🔗

- void **set_pressed**(value: bool)
- bool **is_pressed**()

If `true`, the touch's state is pressed. If `false`, the touch's state is released.

InputEventShortcut

Inherits: InputEvent **<** Resource **<** RefCounted **<** Object

Represents a triggered keyboard Shortcut.

Description

InputEventShortcut is a special event that can be received in Node._input, Node._shortcut_input, and Node._unhandled_input. It is typically sent by the editor's Command Palette to trigger actions, but can also be sent manually using Viewport.push_input.

Properties

Shortcut	shortcut

Property Descriptions

Shortcut **shortcut** 🔗

- void **set_shortcut**(value: Shortcut)
- Shortcut **get_shortcut**()

The Shortcut represented by this event. Its Shortcut.matches_event method will always return `true` for this event.

InputEventWithModifiers

Inherits: InputEventFromWindow **<** InputEvent **<** Resource **<** RefCounted **<** Object

Inherited By: InputEventGesture, InputEventKey, InputEventMouse

Abstract base class for input events affected by modifier keys like `Shift` and `Alt`.

Description

Stores information about mouse, keyboard, and touch gesture input events. This includes information about which modifier keys are pressed, such as `Shift` or `Alt`. See Node._input.

Tutorials

- Using InputEvent

Properties

bool	alt_pressed	`false`
bool	command_or_control_autoremap	`false`
bool	ctrl_pressed	`false`

bool	meta_pressed	`false`
bool	shift_pressed	`false`

Methods

BitField[KeyModifierMask]	get_modifiers_mask() const
bool	is_command_or_control_pressed() const

Property Descriptions

bool **alt_pressed** = `false`

- void **set_alt_pressed**(value: bool)
- bool **is_alt_pressed**()

State of the `Alt` modifier.

bool **command_or_control_autoremap** = `false`

- void **set_command_or_control_autoremap**(value: bool)
- bool **is_command_or_control_autoremap**()

Automatically use `Meta` (`Cmd`) on macOS and `Ctrl` on other platforms. If `true`, ctrl_pressed and meta_pressed cannot be set.

---

bool **ctrl_pressed** = `false`

- void **set_ctrl_pressed**(value: bool)
- bool **is_ctrl_pressed**()

State of the `Ctrl` modifier.

---

bool **meta_pressed** = `false`

- void **set_meta_pressed**(value: bool)
- bool **is_meta_pressed**()

State of the `Meta` modifier. On Windows and Linux, this represents the Windows key (sometimes called "meta" or "super" on Linux). On macOS, this represents the Command key.

---

bool **shift_pressed** = `false`

- void **set_shift_pressed**(value: bool)
- bool **is_shift_pressed**()

State of the `Shift` modifier.

Method Descriptions

BitField[KeyModifierMask] **get_modifiers_mask**() const

Returns the keycode combination of modifier keys.

---

bool **is_command_or_control_pressed**() const

On macOS, returns `true` if `Meta` (`Cmd`) is pressed.

On other platforms, returns `true` if `Ctrl` is pressed.

JSON

Inherits: Resource **<** RefCounted **<** Object

Helper class for creating and parsing JSON data.

Description

The **JSON** class enables all data types to be converted to and from a JSON string. This is useful for serializing data, e.g. to save to a file or send over the network.

stringify is used to convert any data type into a JSON string.

parse is used to convert any existing JSON data into a Variant that can be used within Godot. If successfully parsed, use data to retrieve the Variant, and use `typeof` to check if the Variant's type is what you expect. JSON Objects are converted into a Dictionary, but JSON data can be used to store Arrays, numbers, Strings and even just a boolean.

Example

```
var data_to_send = ["a", "b", "c"]
var json_string = JSON.stringify(data_to_send)
# Save data
# ...
# Retrieve data
var json = JSON.new()
var error = json.parse(json_string)
if error == OK:
    var data_received = json.data
    if typeof(data_received) == TYPE_ARRAY:
        print(data_received) # Prints array
    else:
        print("Unexpected data")
else:
    print("JSON Parse Error: ", json.get_error_message(), " in ", json_string, " at line ", json.get_error_line())
```

Alternatively, you can parse strings using the static parse_string method, but it doesn't handle errors.

```
var data = JSON.parse_string(json_string) # Returns null if parsing failed.
```

Note: Both parse methods do not fully comply with the JSON specification:

- Trailing commas in arrays or objects are ignored, instead of causing a parser error.
- New line and tab characters are accepted in string literals, and are treated like their corresponding escape sequences `\n` and `\t`.
- Numbers are parsed using String.to_float which is generally more lax than the JSON specification.
- Certain errors, such as invalid Unicode sequences, do not cause a parser error. Instead, the string is cleansed and an error is logged to the console.

Properties

Variant	data	`null`

Methods

int	get_error_line() const

String	get_error_message() const
String	get_parsed_text() const
Error	parse(json_text: String, keep_text: bool = false)
Variant	parse_string(json_string: String) static
String	stringify(data: Variant, indent: String = "", sort_keys: bool = true, full_precision: bool = false) static

Property Descriptions

Variant **data** = `null`

- void **set_data**(value: Variant)
- Variant **get_data**()

Contains the parsed JSON data in Variant form.

Method Descriptions

int **get_error_line**() const

Returns `0` if the last call to parse was successful, or the line number where the parse failed.

String **get_error_message**() const

Returns an empty string if the last call to parse was successful, or the error message if it failed.

String **get_parsed_text**() const

Return the text parsed by parse (requires passing `keep_text` to parse).

Error **parse**(json_text: String, keep_text: bool = false)

Attempts to parse the `json_text` provided.

Returns an Error. If the parse was successful, it returns @GlobalScope.OK and the result can be retrieved using data. If unsuccessful, use get_error_line and get_error_message to identify the source of the failure.

Non-static variant of parse_string, if you want custom error handling.

The optional `keep_text` argument instructs the parser to keep a copy of the original text. This text can be obtained later by using the get_parsed_text function and is used when saving the resource (instead of generating new text from data).

Variant **parse_string**(json_string: String) static

Attempts to parse the `json_string` provided and returns the parsed data. Returns `null` if parse failed.

String **stringify**(data: Variant, indent: String = "", sort_keys: bool = true, full_precision: bool = false) static 🔗

Converts a Variant var to JSON text and returns the result. Useful for serializing data to store or send over the network.

Note: The JSON specification does not define integer or float types, but only a *number* type. Therefore, converting a Variant to JSON text will convert all numerical values to float types.

Note: If `full_precision` is `true`, when stringifying floats, the unreliable digits are stringified in addition to the reliable digits to guarantee exact decoding.

The `indent` parameter controls if and how something is indented; its contents will be used where there should be an indent in the output. Even spaces like `"   "` will work. `\t` and `\n` can also be used for a tab indent, or to make a newline for each indent respectively.

Example output:

```
## JSON.stringify(my_dictionary)
{"name":"my_dictionary","version":"1.0.0","entities":
[{"name":"entity_0","value":"value_0"},
{"name":"entity_1","value":"value_1"}]}

## JSON.stringify(my_dictionary, "\t")
{
    "name": "my_dictionary",
    "version": "1.0.0",
    "entities": [
        {
            "name": "entity_0",
            "value": "value_0"
        },
        {
            "name": "entity_1",
            "value": "value_1"
        }
    ]
}

## JSON.stringify(my_dictionary, "...")
{
..."name": "my_dictionary",
..."version": "1.0.0",
..."entities": [
......{
........."name": "entity_0",
........."value": "value_0"
......},
......{
........."name": "entity_1",
........."value": "value_1"
......}
...]
}
```

LabelSettings

Inherits: Resource **<** RefCounted **<** Object

Provides common settings to customize the text in a Label.

Description

LabelSettings is a resource that provides common settings to customize the text in a Label. It will take priority over the properties defined in Control.theme. The resource can be shared between multiple labels and changed on the fly, so it's convenient and flexible way to setup text style.

Properties

Font	font	
Color	font_color	`Color(1, 1, 1, 1)`
int	font_size	`16`
float	line_spacing	`3.0`
Color	outline_color	`Color(1, 1, 1, 1)`

int	outline_size	`0`
Color	shadow_color	`Color(0, 0, 0, 0)`
Vector2	shadow_offset	`Vector2(1, 1)`
int	shadow_size	`1`

Property Descriptions

Font **font** 🔗

- void **set_font**(value: Font)
- Font **get_font**()

Font used for the text.

Color **font_color** = `Color(1, 1, 1, 1)` 🔗

- void **set_font_color**(value: Color)
- Color **get_font_color**()

Color of the text.

---

int **font_size** = `16` 🔗

- void **set_font_size**(value: int)
- int **get_font_size**()

Size of the text.

---

float **line_spacing** = `3.0`

- void **set_line_spacing**(value: float)
- float **get_line_spacing**()

Vertical space between lines when the text is multiline.

Color **outline_color** = `Color(1, 1, 1, 1)`

- void **set_outline_color**(value: Color)
- Color **get_outline_color**()

The color of the outline.

---

int **outline_size** = `0`

- void **set_outline_size**(value: int)
- int **get_outline_size**()

Text outline size.

Color **shadow_color** = `Color(0, 0, 0, 0)`

- void **set_shadow_color**(value: Color)
- Color **get_shadow_color**()

Color of the shadow effect. If alpha is `0`, no shadow will be drawn.

Vector2 **shadow_offset** = `Vector2(1, 1)`

- void **set_shadow_offset**(value: Vector2)
- Vector2 **get_shadow_offset**()

Offset of the shadow effect, in pixels.

---

int **shadow_size** = `1` 🔗

- void **set_shadow_size**(value: int)
- int **get_shadow_size**()

Size of the shadow effect.

LightmapGIData

Inherits: Resource **<** RefCounted **<** Object

Contains baked lightmap and dynamic object probe data for LightmapGI.

Description

LightmapGIData contains baked lightmap and dynamic object probe data for LightmapGI. It is replaced every time lightmaps are baked in LightmapGI.

Properties

TextureLayered	light_texture	
Array[TextureLayered]	lightmap_textures	[ ]

Methods

void	add_user(path: NodePath, uv_scale: Rect2, slice_index: int, sub_instance: int)
void	clear_users()

int	get_user_count() const
NodePath	get_user_path(user_idx: int) const
bool	is_using_spherical_harmonics() const
void	set_uses_spherical_harmonics(uses_spherical_harmonics: bool)

Property Descriptions

TextureLayered **light_texture** 🔗

- void **set_light_texture**(value: TextureLayered)
- TextureLayered **get_light_texture**()

Deprecated: The lightmap atlas can now contain multiple textures. See lightmap_textures.

The lightmap atlas texture generated by the lightmapper.

Array[TextureLayered] **lightmap_textures** = `[]` 🔗

- void **set_lightmap_textures**(value: Array[TextureLayered])
- Array[TextureLayered] **get_lightmap_textures**()

The lightmap atlas textures generated by the lightmapper.

Method Descriptions

void **add_user**(path: NodePath, uv_scale: Rect2, slice_index: int, sub_instance: int)

Adds an object that is considered baked within this **LightmapGIData**.

---

void **clear_users**()

Clear all objects that are considered baked within this **LightmapGIData**.

---

int **get_user_count**() const

Returns the number of objects that are considered baked within this **LightmapGIData**.

NodePath **get_user_path**(user_idx: int) const

Returns the NodePath of the baked object at index `user_idx`.

---

bool **is_using_spherical_harmonics**() const

If `true`, lightmaps were baked with directional information. See also LightmapGI.directional.

---

void **set_uses_spherical_harmonics**(uses_spherical_harmonics: bool)

If `uses_spherical_harmonics` is `true`, tells the engine to treat the lightmap data as if it was baked with directional information.

Note: Changing this value on already baked lightmaps will not cause them to be baked again. This means the material appearance will look incorrect until lightmaps are baked again, in which case the value set here is discarded as the entire **LightmapGIData** resource is replaced by the lightmapper.

Material

Inherits: Resource **<** RefCounted **<** Object

Inherited By: BaseMaterial3D, CanvasItemMaterial, FogMaterial, PanoramaSkyMaterial, ParticleProcessMaterial, PhysicalSkyMaterial, PlaceholderMaterial, ProceduralSkyMaterial, ShaderMaterial

Virtual base class for applying visual properties to an object, such as color and roughness.

Description

Material is a base resource used for coloring and shading geometry. All materials inherit from it and almost all VisualInstance3D derived nodes carry a **Material**. A few flags and parameters are shared between all material types and are configured here.

Importantly, you can inherit from **Material** to create your own custom material type in script or in GDExtension.

Tutorials

- 3D Material Testers Demo [https://godotengine.org/asset-library/asset/2742]
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

Material	next_pass
int	render_priority

Methods

bool	_can_do_next_pass() virtual const
bool	_can_use_render_priority() virtual const
Mode	_get_shader_mode() virtual const
RID	_get_shader_rid() virtual const
Resource	create_placeholder() const
void	inspect_native_shader_code()

Constants

RENDER_PRIORITY_MAX = `127` 🔗

Maximum value for the render_priority parameter.

RENDER_PRIORITY_MIN = `-128` 🔗

Minimum value for the render_priority parameter.

Property Descriptions

Material **next_pass** 🔗

- void **set_next_pass**(value: Material)
- Material **get_next_pass**()

Sets the **Material** to be used for the next pass. This renders the object again using a different material.

Note: next_pass materials are not necessarily drawn immediately after the source **Material**. Draw order is determined by material properties, render_priority, and distance to camera.

Note: This only applies to StandardMaterial3Ds and ShaderMaterials with type "Spatial".

---

int **render_priority** 🔗

- void **set_render_priority**(value: int)
- int **get_render_priority**()

Sets the render priority for objects in 3D scenes. Higher priority objects will be sorted in front of lower priority objects. In other words, all objects with render_priority `1` will render before all objects with render_priority `0`.

Note: This only applies to StandardMaterial3Ds and ShaderMaterials with type "Spatial".

Note: This will not impact how transparent objects are sorted relative to opaque objects or how dynamic meshes will be sorted relative to other opaque meshes. This is because all transparent objects are drawn after all opaque objects and all dynamic opaque meshes are drawn before other opaque meshes.

Method Descriptions

bool **_can_do_next_pass**() virtual const

Only exposed for the purpose of overriding. You cannot call this function directly. Used internally to determine if next_pass should be shown in the editor or not.

---

bool **_can_use_render_priority**() virtual const

Only exposed for the purpose of overriding. You cannot call this function directly. Used internally to determine if render_priority should be shown in the editor or not.

Mode **_get_shader_mode**() virtual const

Only exposed for the purpose of overriding. You cannot call this function directly. Used internally by various editor tools.

RID **_get_shader_rid**() virtual const

Only exposed for the purpose of overriding. You cannot call this function directly. Used internally by various editor tools. Used to access the RID of the **Material**'s Shader.

Resource **create_placeholder**() const

Creates a placeholder version of this resource (PlaceholderMaterial).

---

void **inspect_native_shader_code**() 🔗

Only available when running in the editor. Opens a popup that visualizes the generated shader code, including all variants and internal shader code.

Mesh

Inherits: Resource **<** RefCounted **<** Object

Inherited By: ArrayMesh, ImmediateMesh, PlaceholderMesh, PrimitiveMesh

A Resource that contains vertex array-based geometry.

Description

Mesh is a type of Resource that contains vertex array-based geometry, divided in *surfaces*. Each surface contains a completely separate array and a material used to draw it. Design wise, a mesh with multiple surfaces is preferred to a single surface, because objects created in 3D editing software commonly contain multiple materials. The maximum number of surfaces per mesh is RenderingServer.MAX_MESH_SURFACES.

Tutorials

- 3D Material Testers Demo [https://godotengine.org/asset-library/asset/2742]
- 3D Kinematic Character Demo [https://godotengine.org/asset-library/asset/2739]
- 3D Platformer Demo [https://godotengine.org/asset-library/asset/2748]
- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

Vector2i lightmap_size_hint `Vector2i(0, 0)`

Methods

AABB	_get_aabb() virtual const
int	_get_blend_shape_count() virtual const
StringName	_get_blend_shape_name(index: int) virtual const
int	_get_surface_count() virtual const
void	_set_blend_shape_name(index: int, name: StringName) virtual
int	_surface_get_array_index_len(index: int) virtual const
int	_surface_get_array_len(index: int) virtual const
Array	_surface_get_arrays(index: int) virtual const

Array[Array]	_surface_get_blend_shape_arrays(index: int) virtual const
int	_surface_get_format(index: int) virtual const
Dictionary	_surface_get_lods(index: int) virtual const
Material	_surface_get_material(index: int) virtual const
int	_surface_get_primitive_type(index: int) virtual const
void	_surface_set_material(index: int, material: Material) virtual
ConvexPolygonShape3D	create_convex_shape(clean: bool = true, simplify: bool = false) const
Mesh	create_outline(margin: float) const
Resource	create_placeholder() const
ConcavePolygonShape3D	create_trimesh_shape() const
TriangleMesh	generate_triangle_mesh() const

AABB	get_aabb() const
PackedVector3Array	get_faces() const
int	get_surface_count() const
Array	surface_get_arrays(surf_idx: int) const
Array[Array]	surface_get_blend_shape_arrays(surf_idx: int) const
Material	surface_get_material(surf_idx: int) const
void	surface_set_material(surf_idx: int, material: Material)

Enumerations

enum **PrimitiveType**: 🔗

PrimitiveType **PRIMITIVE_POINTS** = `0`

Render array as points (one vertex equals one point).

PrimitiveType **PRIMITIVE_LINES** = `1`

Render array as lines (every two vertices a line is created).

PrimitiveType **PRIMITIVE_LINE_STRIP** = `2`

Render array as line strip.

PrimitiveType **PRIMITIVE_TRIANGLES** = `3`

Render array as triangles (every three vertices a triangle is created).

PrimitiveType **PRIMITIVE_TRIANGLE_STRIP** = `4`

Render array as triangle strips.

---

enum **ArrayType**: 🔗

ArrayType **ARRAY_VERTEX** = `0`

PackedVector3Array, PackedVector2Array, or Array of vertex positions.

ArrayType **ARRAY_NORMAL** = `1`

PackedVector3Array of vertex normals.

Note: The array has to consist of normal vectors, otherwise they will be normalized by the engine, potentially causing visual discrepancies.

ArrayType **ARRAY_TANGENT** = `2`

PackedFloat32Array of vertex tangents. Each element in groups of 4 floats, first 3 floats determine the tangent, and the last the binormal direction as -1 or 1.

ArrayType **ARRAY_COLOR** = `3`

PackedColorArray of vertex colors.

ArrayType **ARRAY_TEX_UV** = `4`

PackedVector2Array for UV coordinates.

ArrayType **ARRAY_TEX_UV2** = `5`

PackedVector2Array for second UV coordinates.

ArrayType **ARRAY_CUSTOM0** = `6`

Contains custom color channel 0. PackedByteArray if `(format >> Mesh.ARRAY_FORMAT_CUSTOM0_SHIFT) & Mesh.ARRAY_FORMAT_CUSTOM_MASK` is ARRAY_CUSTOM_RGBA8_UNORM, ARRAY_CUSTOM_RGBA8_SNORM, ARRAY_CUSTOM_RG_HALF, or ARRAY_CUSTOM_RGBA_HALF. PackedFloat32Array otherwise.

ArrayType **ARRAY_CUSTOM1** = `7`

Contains custom color channel 1. PackedByteArray if `(format >> Mesh.ARRAY_FORMAT_CUSTOM1_SHIFT) & Mesh.ARRAY_FORMAT_CUSTOM_MASK` is ARRAY_CUSTOM_RGBA8_UNORM, ARRAY_CUSTOM_RGBA8_SNORM, ARRAY_CUSTOM_RG_HALF, or ARRAY_CUSTOM_RGBA_HALF. PackedFloat32Array otherwise.

ArrayType **ARRAY_CUSTOM2** = `8`

Contains custom color channel 2. PackedByteArray if `(format >> Mesh.ARRAY_FORMAT_CUSTOM2_SHIFT) & Mesh.ARRAY_FORMAT_CUSTOM_MASK` is ARRAY_CUSTOM_RGBA8_UNORM, ARRAY_CUSTOM_RGBA8_SNORM, ARRAY_CUSTOM_RG_HALF, or ARRAY_CUSTOM_RGBA_HALF. PackedFloat32Array otherwise.

ArrayType **ARRAY_CUSTOM3** = `9`

Contains custom color channel 3. PackedByteArray if `(format >> Mesh.ARRAY_FORMAT_CUSTOM3_SHIFT) & Mesh.ARRAY_FORMAT_CUSTOM_MASK` is ARRAY_CUSTOM_RGBA8_UNORM, ARRAY_CUSTOM_RGBA8_SNORM, ARRAY_CUSTOM_RG_HALF, or ARRAY_CUSTOM_RGBA_HALF. PackedFloat32Array otherwise.

ArrayType **ARRAY_BONES** = `10`

PackedFloat32Array or PackedInt32Array of bone indices. Contains either 4 or 8 numbers per vertex depending on the presence of the ARRAY_FLAG_USE_8_BONE_WEIGHTS flag.

ArrayType **ARRAY_WEIGHTS** = `11`

PackedFloat32Array or PackedFloat64Array of bone weights in the range `0.0` to `1.0` (inclusive). Contains either 4 or 8 numbers per vertex depending on the presence of the ARRAY_FLAG_USE_8_BONE_WEIGHTS flag.

ArrayType **ARRAY_INDEX** = `12`

PackedInt32Array of integers used as indices referencing vertices, colors, normals, tangents, and textures. All of those arrays must have the same number of elements as the vertex array. No index can be beyond the vertex array size. When this index array is present, it puts the function into "index mode," where the index selects the *i*th vertex, normal, tangent, color, UV, etc. This means if you want to have different normals or colors along an edge, you have to duplicate the vertices.

For triangles, the index array is interpreted as triples, referring to the vertices of each triangle. For lines, the index array is in pairs indicating the start and end of each line.

ArrayType **ARRAY_MAX** = `13`

Represents the size of the ArrayType enum.

---

enum **ArrayCustomFormat**: 🔗

ArrayCustomFormat **ARRAY_CUSTOM_RGBA8_UNORM** = `0`

Indicates this custom channel contains unsigned normalized byte colors from 0 to 1, encoded as PackedByteArray.

ArrayCustomFormat **ARRAY_CUSTOM_RGBA8_SNORM** = `1`

Indicates this custom channel contains signed normalized byte colors from -1 to 1, encoded as PackedByteArray.

ArrayCustomFormat **ARRAY_CUSTOM_RG_HALF** = `2`

Indicates this custom channel contains half precision float colors, encoded as PackedByteArray. Only red and green channels are used.

ArrayCustomFormat **ARRAY_CUSTOM_RGBA_HALF** = `3`

Indicates this custom channel contains half precision float colors, encoded as PackedByteArray.

ArrayCustomFormat **ARRAY_CUSTOM_R_FLOAT** = `4`

Indicates this custom channel contains full float colors, in a PackedFloat32Array. Only the red channel is used.

ArrayCustomFormat **ARRAY_CUSTOM_RG_FLOAT** = `5`

Indicates this custom channel contains full float colors, in a PackedFloat32Array. Only red and green channels are used.

ArrayCustomFormat **ARRAY_CUSTOM_RGB_FLOAT** = `6`

Indicates this custom channel contains full float colors, in a PackedFloat32Array. Only red, green and blue channels are used.

ArrayCustomFormat **ARRAY_CUSTOM_RGBA_FLOAT** = `7`

Indicates this custom channel contains full float colors, in a PackedFloat32Array.

ArrayCustomFormat **ARRAY_CUSTOM_MAX** = `8`

Represents the size of the ArrayCustomFormat enum.

---

flags **ArrayFormat**: 🔗

ArrayFormat **ARRAY_FORMAT_VERTEX** = `1`

Mesh array contains vertices. All meshes require a vertex array so this should always be present.

ArrayFormat **ARRAY_FORMAT_NORMAL** = `2`

Mesh array contains normals.

ArrayFormat **ARRAY_FORMAT_TANGENT** = `4`

Mesh array contains tangents.

ArrayFormat **ARRAY_FORMAT_COLOR** = `8`

Mesh array contains colors.

ArrayFormat **ARRAY_FORMAT_TEX_UV** = `16`

Mesh array contains UVs.

ArrayFormat **ARRAY_FORMAT_TEX_UV2** = `32`

Mesh array contains second UV.

ArrayFormat **ARRAY_FORMAT_CUSTOM0** = `64`

Mesh array contains custom channel index 0.

ArrayFormat **ARRAY_FORMAT_CUSTOM1** = `128`

Mesh array contains custom channel index 1.

ArrayFormat **ARRAY_FORMAT_CUSTOM2** = `256`

Mesh array contains custom channel index 2.

ArrayFormat **ARRAY_FORMAT_CUSTOM3** = `512`

Mesh array contains custom channel index 3.

ArrayFormat **ARRAY_FORMAT_BONES** = `1024`

Mesh array contains bones.

ArrayFormat **ARRAY_FORMAT_WEIGHTS** = `2048`

Mesh array contains bone weights.

ArrayFormat **ARRAY_FORMAT_INDEX** = `4096`

Mesh array uses indices.

ArrayFormat **ARRAY_FORMAT_BLEND_SHAPE_MASK** = `7`

Mask of mesh channels permitted in blend shapes.

ArrayFormat **ARRAY_FORMAT_CUSTOM_BASE** = `13`

Shift of first custom channel.

ArrayFormat **ARRAY_FORMAT_CUSTOM_BITS** = `3`

Number of format bits per custom channel. See ArrayCustomFormat.

ArrayFormat **ARRAY_FORMAT_CUSTOM0_SHIFT** = `13`

Amount to shift ArrayCustomFormat for custom channel index 0.

ArrayFormat **ARRAY_FORMAT_CUSTOM1_SHIFT** = `16`

Amount to shift ArrayCustomFormat for custom channel index 1.

ArrayFormat **ARRAY_FORMAT_CUSTOM2_SHIFT** = `19`

Amount to shift ArrayCustomFormat for custom channel index 2.

ArrayFormat **ARRAY_FORMAT_CUSTOM3_SHIFT** = `22`

Amount to shift ArrayCustomFormat for custom channel index 3.

ArrayFormat **ARRAY_FORMAT_CUSTOM_MASK** = `7`

Mask of custom format bits per custom channel. Must be shifted by one of the SHIFT constants. See ArrayCustomFormat.

ArrayFormat **ARRAY_COMPRESS_FLAGS_BASE** = `25`

Shift of first compress flag. Compress flags should be passed to ArrayMesh.add_surface_from_arrays and SurfaceTool.commit.

ArrayFormat **ARRAY_FLAG_USE_2D_VERTICES** = `33554432`

Flag used to mark that the array contains 2D vertices.

ArrayFormat **ARRAY_FLAG_USE_DYNAMIC_UPDATE** = `67108864`

Flag indices that the mesh data will use `GL_DYNAMIC_DRAW` on GLES. Unused on Vulkan.

ArrayFormat **ARRAY_FLAG_USE_8_BONE_WEIGHTS** = `134217728`

Flag used to mark that the mesh contains up to 8 bone influences per vertex. This flag indicates that ARRAY_BONES and ARRAY_WEIGHTS elements will have double length.

ArrayFormat **ARRAY_FLAG_USES_EMPTY_VERTEX_ARRAY** = `268435456`

Flag used to mark that the mesh intentionally contains no vertex array.

ArrayFormat **ARRAY_FLAG_COMPRESS_ATTRIBUTES** = `536870912`

Flag used to mark that a mesh is using compressed attributes (vertices, normals, tangents, UVs). When this form of compression is enabled, vertex positions will be packed into an RGBA16UNORM attribute and scaled in the vertex shader. The normal and tangent will be packed into an RG16UNORM representing an axis, and a 16-bit float stored in the A-channel of the vertex. UVs will use 16-bit normalized floats instead of full 32-bit signed floats. When using this compression mode you must use either vertices, normals, and tangents or only vertices. You cannot use normals without tangents. Importers will automatically enable this compression if they can.

enum **BlendShapeMode**: 🔗

BlendShapeMode **BLEND_SHAPE_MODE_NORMALIZED** = `0`

Blend shapes are normalized.

BlendShapeMode **BLEND_SHAPE_MODE_RELATIVE** = `1`

Blend shapes are relative to base weight.

Property Descriptions

Vector2i **lightmap_size_hint** = `Vector2i(0, 0)` 🔗

- void **set_lightmap_size_hint**(value: Vector2i)
- Vector2i **get_lightmap_size_hint**()

Sets a hint to be used for lightmap resolution.

Method Descriptions

AABB **_get_aabb**() virtual const 🔗

Virtual method to override the AABB for a custom class extending **Mesh**.

---

int **_get_blend_shape_count**() virtual const 🔗

Virtual method to override the number of blend shapes for a custom class extending **Mesh**.

StringName **_get_blend_shape_name**(index: int) virtual const 🔗

Virtual method to override the retrieval of blend shape names for a custom class extending **Mesh**.

---

int **_get_surface_count**() virtual const 🔗

Virtual method to override the surface count for a custom class extending **Mesh**.

---

void **_set_blend_shape_name**(index: int, name: StringName) virtual 🔗

Virtual method to override the names of blend shapes for a custom class extending **Mesh**.

---

int **_surface_get_array_index_len**(index: int) virtual const 🔗

Virtual method to override the surface array index length for a custom class extending **Mesh**.

---

int **_surface_get_array_len**(index: int) virtual const 🔗

Virtual method to override the surface array length for a custom class extending **Mesh**.

Array **_surface_get_arrays**(index: int) virtual const 🔗

Virtual method to override the surface arrays for a custom class extending **Mesh**.

Array[Array] **_surface_get_blend_shape_arrays**(index: int) virtual const 🔗

Virtual method to override the blend shape arrays for a custom class extending **Mesh**.

---

int **_surface_get_format**(index: int) virtual const 🔗

Virtual method to override the surface format for a custom class extending **Mesh**.

Dictionary **_surface_get_lods**(index: int) virtual const 🔗

Virtual method to override the surface LODs for a custom class extending **Mesh**.

Material **_surface_get_material**(index: int) virtual const

Virtual method to override the surface material for a custom class extending **Mesh**.

---

int **_surface_get_primitive_type**(index: int) virtual const

Virtual method to override the surface primitive type for a custom class extending **Mesh**.

---

void **_surface_set_material**(index: int, material: Material) virtual

Virtual method to override the setting of a `material` at the given `index` for a custom class extending **Mesh**.

ConvexPolygonShape3D **create_convex_shape**(clean: bool = true, simplify: bool = false) const

Calculate a ConvexPolygonShape3D from the mesh.

If `clean` is `true` (default), duplicate and interior vertices are removed automatically. You can set it to `false` to make the process faster if not needed.

If `simplify` is `true`, the geometry can be further simplified to reduce the number of vertices. Disabled by default.

Mesh **create_outline**(margin: float) const

Calculate an outline mesh at a defined offset (margin) from the original mesh.

Note: This method typically returns the vertices in reverse order (e.g. clockwise to counterclockwise).

Resource **create_placeholder**() const

Creates a placeholder version of this resource (PlaceholderMesh).

ConcavePolygonShape3D **create_trimesh_shape**() const

Calculate a ConcavePolygonShape3D from the mesh.

TriangleMesh **generate_triangle_mesh**() const

Generate a TriangleMesh from the mesh. Considers only surfaces using one of these primitive types: PRIMITIVE_TRIANGLES, PRIMITIVE_TRIANGLE_STRIP.

AABB **get_aabb**() const

Returns the smallest AABB enclosing this mesh in local space. Not affected by `custom_aabb`.

Note: This is only implemented for ArrayMesh and PrimitiveMesh.

PackedVector3Array **get_faces**() const

Returns all the vertices that make up the faces of the mesh. Each three vertices represent one triangle.

int **get_surface_count**() const

Returns the number of surfaces that the **Mesh** holds. This is equivalent to MeshInstance3D.get_surface_override_material_count.

Array **surface_get_arrays**(surf_idx: int) const

Returns the arrays for the vertices, normals, UVs, etc. that make up the requested surface (see ArrayMesh.add_surface_from_arrays).

Array[Array] **surface_get_blend_shape_arrays**(surf_idx: int) const

Returns the blend shape arrays for the requested surface.

Material **surface_get_material**(surf_idx: int) const

Returns a Material in a given surface. Surface is rendered using this material.

Note: This returns the material within the **Mesh** resource, not the Material associated to the MeshInstance3D's Surface Material Override properties. To get the Material associated to the MeshInstance3D's Surface Material Override properties, use MeshInstance3D.get_surface_override_material instead.

---

void **surface_set_material**(surf_idx: int, material: Material)

Sets a Material for a given surface. Surface will be rendered using this material.

Note: This assigns the material within the **Mesh** resource, not the Material associated to the MeshInstance3D's Surface Material Override properties. To set the Material associated to the

MeshInstance3D's Surface Material Override properties, use MeshInstance3D.set_surface_override_material instead.

MeshLibrary

Inherits: Resource **<** RefCounted **<** Object

Library of meshes.

Description

A library of meshes. Contains a list of Mesh resources, each with a name and ID. Each item can also include collision and navigation shapes. This resource is used in GridMap.

Tutorials

- 3D Kinematic Character Demo [https://godotengine.org/asset-library/asset/2739]
- 3D Platformer Demo [https://godotengine.org/asset-library/asset/2748]

Methods

void	clear()
void	create_item(id: int)
int	find_item_by_name(name: String) const
PackedInt32Array	get_item_list() const

Mesh	get_item_mesh(id: int) const
Transform3D	get_item_mesh_transform(id: int) const
String	get_item_name(id: int) const
int	get_item_navigation_layers(id: int) const
NavigationMesh	get_item_navigation_mesh(id: int) const
Transform3D	get_item_navigation_mesh_transform(id: int) const
Texture2D	get_item_preview(id: int) const
Array	get_item_shapes(id: int) const
int	get_last_unused_item_id() const
void	remove_item(id: int)
void	set_item_mesh(id: int, mesh: Mesh)
void	set_item_mesh_transform(id: int, mesh_transform: Transform3D)

void	set_item_name(id: int, name: String)
void	set_item_navigation_layers(id: int, navigation_layers: int)
void	set_item_navigation_mesh(id: int, navigation_mesh: NavigationMesh)
void	set_item_navigation_mesh_transform(id: int, navigation_mesh: Transform3D)
void	set_item_preview(id: int, texture: Texture2D)
void	set_item_shapes(id: int, shapes: Array)

Method Descriptions

void **clear**() 🔗

Clears the library.

---

void **create_item**(id: int) 🔗

Creates a new item in the library with the given ID.

You can get an unused ID from get_last_unused_item_id.

int **find_item_by_name**(name: String) const 🔗

Returns the first item with the given name, or `-1` if no item is found.

PackedInt32Array **get_item_list**() const 🔗

Returns the list of item IDs in use.

Mesh **get_item_mesh**(id: int) const 🔗

Returns the item's mesh.

Transform3D **get_item_mesh_transform**(id: int) const 🔗

Returns the transform applied to the item's mesh.

String **get_item_name**(id: int) const 🔗

Returns the item's name.

---

int **get_item_navigation_layers**(id: int) const 🔗

Returns the item's navigation layers bitmask.

NavigationMesh **get_item_navigation_mesh**(id: int) const 🔗

Returns the item's navigation mesh.

Transform3D **get_item_navigation_mesh_transform**(id: int) const 🔗

Returns the transform applied to the item's navigation mesh.

Texture2D **get_item_preview**(id: int) const

When running in the editor, returns a generated item preview (a 3D rendering in isometric perspective). When used in a running project, returns the manually-defined item preview which can be set using set_item_preview. Returns an empty Texture2D if no preview was manually set in a running project.

Array **get_item_shapes**(id: int) const

Returns an item's collision shapes.

The array consists of each Shape3D followed by its Transform3D.

int **get_last_unused_item_id**() const

Gets an unused ID for a new item.

void **remove_item**(id: int)

Removes the item.

void **set_item_mesh**(id: int, mesh: Mesh)

Sets the item's mesh.

void **set_item_mesh_transform**(id: int, mesh_transform: Transform3D)

Sets the transform to apply to the item's mesh.

void **set_item_name**(id: int, name: String)

Sets the item's name.

This name is shown in the editor. It can also be used to look up the item later using find_item_by_name.

void **set_item_navigation_layers**(id: int, navigation_layers: int)

Sets the item's navigation layers bitmask.

void **set_item_navigation_mesh**(id: int, navigation_mesh: NavigationMesh)

Sets the item's navigation mesh.

void **set_item_navigation_mesh_transform**(id: int, navigation_mesh: Transform3D)

Sets the transform to apply to the item's navigation mesh.

void **set_item_preview**(id: int, texture: Texture2D)

Sets a texture to use as the item's preview icon in the editor.

void **set_item_shapes**(id: int, shapes: Array)

Sets an item's collision shapes.

The array should consist of Shape3D objects, each followed by a Transform3D that will be applied to it. For shapes that should not have a transform, use Transform3D.IDENTITY.

MeshTexture

Inherits: Texture2D **<** Texture **<** Resource **<** RefCounted **<** Object

Simple texture that uses a mesh to draw itself.

Description

Simple texture that uses a mesh to draw itself. It's limited because flags can't be changed and region drawing is not supported.

Properties

Texture2D	base_texture	
Vector2	image_size	`Vector2(0, 0)`
Mesh	mesh	
bool	resource_local_to_scene	`false` (overrides Resource)

Property Descriptions

Texture2D **base_texture** 🔗

- void **set_base_texture**(value: Texture2D)

- Texture2D **get_base_texture**()

Sets the base texture that the Mesh will use to draw.

Vector2 **image_size** = `Vector2(0, 0)` 🔗

- void **set_image_size**(value: Vector2)
- Vector2 **get_image_size**()

Sets the size of the image, needed for reference.

Mesh **mesh** 🔗

- void **set_mesh**(value: Mesh)
- Mesh **get_mesh**()

Sets the mesh used to draw. It must be a mesh using 2D vertices.

MissingResource

Inherits: Resource **<** RefCounted **<** Object

An internal editor class intended for keeping the data of unrecognized resources.

Description

This is an internal editor class intended for keeping data of resources of unknown type (most likely this type was supplied by an extension that is no longer loaded). It can't be manually instantiated or placed in a scene.

Warning: Ignore missing resources unless you know what you are doing. Existing properties on a missing resource can be freely modified in code, regardless of the type they are intended to be.

Properties

String	original_class
bool	recording_properties

Property Descriptions

String **original_class** 🔗

- void **set_original_class**(value: String)
- String **get_original_class**()

The name of the class this resource was supposed to be (see Object.get_class).

---

bool **recording_properties**

- void **set_recording_properties**(value: bool)
- bool **is_recording_properties**()

If set to `true`, allows new properties to be added on top of the existing ones with Object.set.

MultiMesh

Inherits: Resource **<** RefCounted **<** Object

Provides high-performance drawing of a mesh multiple times using GPU instancing.

Description

MultiMesh provides low-level mesh instancing. Drawing thousands of MeshInstance3D nodes can be slow, since each object is submitted to the GPU then drawn individually.

MultiMesh is much faster as it can draw thousands of instances with a single draw call, resulting in less API overhead.

As a drawback, if the instances are too far away from each other, performance may be reduced as every single instance will always render (they are spatially indexed as one, for the whole object).

Since instances may have any behavior, the AABB used for visibility must be provided by the user.

Note: A MultiMesh is a single object, therefore the same maximum lights per object restriction applies. This means, that once the maximum lights are consumed by one or more instances, the rest of the MultiMesh instances will **not** receive any lighting.

Note: Blend Shapes will be ignored if used in a MultiMesh.

Tutorials

- Using MultiMeshInstance
- Optimization using MultiMeshes
- Animating thousands of fish with MultiMeshInstance

Properties

PackedFloat32Array	buffer	`PackedFloat32Array()`
PackedColorArray	color_array	
AABB	custom_aabb	`AABB(0, 0, 0, 0, 0, 0)`
PackedColorArray	custom_data_array	
int	instance_count	`0`
Mesh	mesh	
PackedVector2Array	transform_2d_array	
PackedVector3Array	transform_array	
TransformFormat	transform_format	`0`
bool	use_colors	`false`

bool	use_custom_data	`false`
int	visible_instance_count	`-1`

Methods

AABB	get_aabb() const
Color	get_instance_color(instance: int) const
Color	get_instance_custom_data(instance: int) const
Transform3D	get_instance_transform(instance: int) const
Transform2D	get_instance_transform_2d(instance: int) const
void	set_instance_color(instance: int, color: Color)
void	set_instance_custom_data(instance: int, custom_data: Color)
void	set_instance_transform(instance: int, transform: Transform3D)

void	set_instance_transform_2d(instance: int, transform: Transform2D)

Enumerations

enum **TransformFormat**:

TransformFormat **TRANSFORM_2D** = `0`

Use this when using 2D transforms.

TransformFormat **TRANSFORM_3D** = `1`

Use this when using 3D transforms.

Property Descriptions

PackedFloat32Array **buffer** = `PackedFloat32Array()`

- void **set_buffer**(value: PackedFloat32Array)
- PackedFloat32Array **get_buffer**()

There is currently no description for this property. Please help us by contributing one!

PackedColorArray **color_array**

Deprecated: Accessing this property is very slow. Use set_instance_color and get_instance_color instead.

Array containing each Color used by all instances of this mesh.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedColorArray for more details.

AABB **custom_aabb** = `AABB(0, 0, 0, 0, 0, 0)` 🔗

- void **set_custom_aabb**(value: AABB)
- AABB **get_custom_aabb**()

Custom AABB for this MultiMesh resource. Setting this manually prevents costly runtime AABB recalculations.

PackedColorArray **custom_data_array** 🔗

Deprecated: Accessing this property is very slow. Use set_instance_custom_data and get_instance_custom_data instead.

Array containing each custom data value used by all instances of this mesh, as a PackedColorArray.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedColorArray for more details.

---

int **instance_count** = `0` 🔗

- void **set_instance_count**(value: int)
- int **get_instance_count**()

Number of instances that will get drawn. This clears and (re)sizes the buffers. Setting data format or flags afterwards will have no effect.

By default, all instances are drawn but you can limit this with visible_instance_count.

Mesh **mesh** 🔗

- void **set_mesh**(value: Mesh)
- Mesh **get_mesh**()

Mesh resource to be instanced.

The looks of the individual instances can be modified using set_instance_color and set_instance_custom_data.

PackedVector2Array **transform_2d_array** 🔗

Deprecated: Accessing this property is very slow. Use set_instance_transform_2d and get_instance_transform_2d instead.

Array containing each Transform2D value used by all instances of this mesh, as a PackedVector2Array. Each transform is divided into 3 Vector2 values corresponding to the transforms' `x`, `y`, and `origin`.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector2Array for more details.

PackedVector3Array **transform_array** 🔗

Deprecated: Accessing this property is very slow. Use set_instance_transform and get_instance_transform instead.

Array containing each Transform3D value used by all instances of this mesh, as a PackedVector3Array. Each transform is divided into 4 Vector3 values corresponding to the transforms' `x`, `y`, `z`, and `origin`.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector3Array for more details.

TransformFormat **transform_format** = `0` 🔗

- void **set_transform_format**(value: TransformFormat)
- TransformFormat **get_transform_format**()

Format of transform used to transform mesh, either 2D or 3D.

---

bool **use_colors** = `false` 🔗

- void **set_use_colors**(value: bool)
- bool **is_using_colors**()

If `true`, the **MultiMesh** will use color data (see set_instance_color). Can only be set when instance_count is `0` or less. This means that you need to call this method before setting the instance count, or temporarily reset it to `0`.

---

bool **use_custom_data** = `false` 🔗

- void **set_use_custom_data**(value: bool)
- bool **is_using_custom_data**()

If `true`, the **MultiMesh** will use custom data (see set_instance_custom_data). Can only be set when instance_count is `0` or less. This means that you need to call this method before setting the instance count, or temporarily reset it to `0`.

---

int **visible_instance_count** = `-1` 🔗

- void **set_visible_instance_count**(value: int)
- int **get_visible_instance_count**()

Limits the number of instances drawn, -1 draws all instances. Changing this does not change the sizes of the buffers.

Method Descriptions

AABB **get_aabb**() const

Returns the visibility axis-aligned bounding box in local space.

Color **get_instance_color**(instance: int) const

Gets a specific instance's color multiplier.

Color **get_instance_custom_data**(instance: int) const

Returns the custom data that has been set for a specific instance.

Transform3D **get_instance_transform**(instance: int) const

Returns the Transform3D of a specific instance.

Transform2D **get_instance_transform_2d**(instance: int) const

Returns the Transform2D of a specific instance.

void **set_instance_color**(instance: int, color: Color)

Sets the color of a specific instance by *multiplying* the mesh's existing vertex colors. This allows for different color tinting per instance.

Note: Each component is stored in 32 bits in the Forward+ and Mobile rendering methods, but is packed into 16 bits in the Compatibility rendering method.

For the color to take effect, ensure that use_colors is `true` on the **MultiMesh** and BaseMaterial3D.vertex_color_use_as_albedo is `true` on the material. If you intend to set an absolute color instead of tinting, make sure the material's albedo color is set to pure white (`Color(1, 1, 1)`).

---

void **set_instance_custom_data**(instance: int, custom_data: Color) 🔗

Sets custom data for a specific instance. `custom_data` is a Color type only to contain 4 floating-point numbers.

Note: Each number is stored in 32 bits in the Forward+ and Mobile rendering methods, but is packed into 16 bits in the Compatibility rendering method.

For the custom data to be used, ensure that use_custom_data is `true`.

This custom instance data has to be manually accessed in your custom shader using `INSTANCE_CUSTOM`.

---

void **set_instance_transform**(instance: int, transform: Transform3D) 🔗

Sets the Transform3D for a specific instance.

---

void **set_instance_transform_2d**(instance: int, transform: Transform2D)

Sets the Transform2D for a specific instance.

NavigationMesh

Experimental: This class may be changed or removed in future versions.

Inherits: Resource **<** RefCounted **<** Object

A navigation mesh that defines traversable areas and obstacles.

Description

A navigation mesh is a collection of polygons that define which areas of an environment are traversable to aid agents in pathfinding through complicated spaces.

Tutorials

- Using NavigationMeshes
- 3D Navigation Demo [https://godotengine.org/asset-library/asset/2743]

Properties

float	agent_height	`1.5`
float	agent_max_climb	`0.25`
float	agent_max_slope	`45.0`
float	agent_radius	`0.5`
float	border_size	`0.0`
float	cell_height	`0.25`
float	cell_size	`0.25`
float	detail_sample_distance	`6.0`

float	detail_sample_max_error	`1.0`
float	edge_max_error	`1.3`
float	edge_max_length	`0.0`
AABB	filter_baking_aabb	`AABB(0, 0, 0, 0, 0, 0)`
Vector3	filter_baking_aabb_offset	`Vector3(0, 0, 0)`
bool	filter_ledge_spans	`false`
bool	filter_low_hanging_obstacles	`false`
bool	filter_walkable_low_height_spans	`false`
int	geometry_collision_mask	`4294967295`
ParsedGeometryType	geometry_parsed_geometry_type	`0`
SourceGeometryMode	geometry_source_geometry_mode	`0`
StringName	geometry_source_group_name	`&"navigation_mesh_source_group"`
float	region_merge_size	`20.0`
float	region_min_size	`2.0`
SamplePartitionType	sample_partition_type	`0`
float	vertices_per_polygon	`6.0`

Methods

void	add_polygon(polygon: PackedInt32Array)
void	clear()
void	clear_polygons()
void	create_from_mesh(mesh: Mesh)
bool	get_collision_mask_value(layer_number: int) const
PackedInt32Array	get_polygon(idx: int)
int	get_polygon_count() const
PackedVector3Array	get_vertices() const
void	set_collision_mask_value(layer_number: int, value: bool)
void	set_vertices(vertices: PackedVector3Array)

Enumerations

enum **SamplePartitionType**: 🔗

SamplePartitionType **SAMPLE_PARTITION_WATERSHED** = `0`

Watershed partitioning. Generally the best choice if you precompute the navigation mesh, use this if you have large open areas.

SamplePartitionType **SAMPLE_PARTITION_MONOTONE** = `1`

Monotone partitioning. Use this if you want fast navigation mesh generation.

SamplePartitionType **SAMPLE_PARTITION_LAYERS** = `2`

Layer partitioning. Good choice to use for tiled navigation mesh with medium and small sized tiles.

SamplePartitionType **SAMPLE_PARTITION_MAX** = `3`

Represents the size of the SamplePartitionType enum.

---

enum **ParsedGeometryType**: 🔗

ParsedGeometryType **PARSED_GEOMETRY_MESH_INSTANCES** = `0`

Parses mesh instances as geometry. This includes MeshInstance3D, CSGShape3D, and GridMap nodes.

ParsedGeometryType **PARSED_GEOMETRY_STATIC_COLLIDERS** = `1`

Parses StaticBody3D colliders as geometry. The collider should be in any of the layers specified by geometry_collision_mask.

ParsedGeometryType **PARSED_GEOMETRY_BOTH** = `2`

Both PARSED_GEOMETRY_MESH_INSTANCES and PARSED_GEOMETRY_STATIC_COLLIDERS.

ParsedGeometryType **PARSED_GEOMETRY_MAX** = `3`

Represents the size of the ParsedGeometryType enum.

---

enum **SourceGeometryMode**: 🔗

SourceGeometryMode **SOURCE_GEOMETRY_ROOT_NODE_CHILDREN** = `0`

Scans the child nodes of the root node recursively for geometry.

SourceGeometryMode **SOURCE_GEOMETRY_GROUPS_WITH_CHILDREN** = `1`

Scans nodes in a group and their child nodes recursively for geometry. The group is specified by geometry_source_group_name.

SourceGeometryMode **SOURCE_GEOMETRY_GROUPS_EXPLICIT** = `2`

Uses nodes in a group for geometry. The group is specified by geometry_source_group_name.

SourceGeometryMode **SOURCE_GEOMETRY_MAX** = `3`

Represents the size of the SourceGeometryMode enum.

Property Descriptions

float **agent_height** = `1.5` 🔗

- void **set_agent_height**(value: float)

- float **get_agent_height**()

The minimum floor to ceiling height that will still allow the floor area to be considered walkable.

Note: While baking, this value will be rounded up to the nearest multiple of cell_height.

---

float **agent_max_climb** = `0.25` 🔗

- void **set_agent_max_climb**(value: float)
- float **get_agent_max_climb**()

The minimum ledge height that is considered to still be traversable.

Note: While baking, this value will be rounded down to the nearest multiple of cell_height.

---

float **agent_max_slope** = `45.0` 🔗

- void **set_agent_max_slope**(value: float)
- float **get_agent_max_slope**()

The maximum slope that is considered walkable, in degrees.

---

float **agent_radius** = `0.5` 🔗

- void **set_agent_radius**(value: float)
- float **get_agent_radius**()

The distance to erode/shrink the walkable area of the heightfield away from obstructions.

Note: While baking, this value will be rounded up to the nearest multiple of cell_size.

---

float **border_size** = `0.0` 🔗

- void **set_border_size**(value: float)
- float **get_border_size**()

The size of the non-navigable border around the bake bounding area.

In conjunction with the filter_baking_aabb and a edge_max_error value at `1.0` or below the border size can be used to bake tile aligned navigation meshes without the tile edges being shrunk by agent_radius.

Note: While baking and not zero, this value will be rounded up to the nearest multiple of cell_size.

---

float **cell_height** = `0.25`

- void **set_cell_height**(value: float)
- float **get_cell_height**()

The cell height used to rasterize the navigation mesh vertices on the Y axis. Must match with the cell height on the navigation map.

---

float **cell_size** = `0.25`

- void **set_cell_size**(value: float)
- float **get_cell_size**()

The cell size used to rasterize the navigation mesh vertices on the XZ plane. Must match with the cell size on the navigation map.

---

float **detail_sample_distance** = `6.0`

- void **set_detail_sample_distance**(value: float)
- float **get_detail_sample_distance**()

The sampling distance to use when generating the detail mesh, in cell unit.

---

float **detail_sample_max_error** = `1.0`

- void **set_detail_sample_max_error**(value: float)
- float **get_detail_sample_max_error**()

The maximum distance the detail mesh surface should deviate from heightfield, in cell unit.

---

float **edge_max_error** = `1.3`

- void **set_edge_max_error**(value: float)
- float **get_edge_max_error**()

The maximum distance a simplified contour's border edges should deviate the original raw contour.

---

float **edge_max_length** = `0.0`

- void **set_edge_max_length**(value: float)
- float **get_edge_max_length**()

The maximum allowed length for contour edges along the border of the mesh. A value of `0.0` disables this feature.

Note: While baking, this value will be rounded up to the nearest multiple of cell_size.

AABB **filter_baking_aabb** = `AABB(0, 0, 0, 0, 0, 0)`

- void **set_filter_baking_aabb**(value: AABB)
- AABB **get_filter_baking_aabb**()

If the baking AABB has a volume the navigation mesh baking will be restricted to its enclosing area.

Vector3 **filter_baking_aabb_offset** = `Vector3(0, 0, 0)`

- void **set_filter_baking_aabb_offset**(value: Vector3)
- Vector3 **get_filter_baking_aabb_offset**()

The position offset applied to the filter_baking_aabb AABB.

---

bool **filter_ledge_spans** = `false`

- void **set_filter_ledge_spans**(value: bool)
- bool **get_filter_ledge_spans**()

If `true`, marks spans that are ledges as non-walkable.

---

bool **filter_low_hanging_obstacles** = `false`

- void **set_filter_low_hanging_obstacles**(value: bool)
- bool **get_filter_low_hanging_obstacles**()

If `true`, marks non-walkable spans as walkable if their maximum is within agent_max_climb of a walkable neighbor.

---

bool **filter_walkable_low_height_spans** = `false`

- void **set_filter_walkable_low_height_spans**(value: bool)
- bool **get_filter_walkable_low_height_spans**()

If `true`, marks walkable spans as not walkable if the clearance above the span is less than agent_height.

---

int **geometry_collision_mask** = `4294967295`

- void **set_collision_mask**(value: int)
- int **get_collision_mask**()

The physics layers to scan for static colliders.

Only used when geometry_parsed_geometry_type is PARSED_GEOMETRY_STATIC_COLLIDERS or PARSED_GEOMETRY_BOTH.

ParsedGeometryType **geometry_parsed_geometry_type** = `0`

- void **set_parsed_geometry_type**(value: ParsedGeometryType)
- ParsedGeometryType **get_parsed_geometry_type**()

Determines which type of nodes will be parsed as geometry. See ParsedGeometryType for possible values.

SourceGeometryMode **geometry_source_geometry_mode** = `0`

- void **set_source_geometry_mode**(value: SourceGeometryMode)
- SourceGeometryMode **get_source_geometry_mode**()

The source of the geometry used when baking. See SourceGeometryMode for possible values.

StringName **geometry_source_group_name** = `&"navigation_mesh_source_group"`

- void **set_source_group_name**(value: StringName)
- StringName **get_source_group_name**()

The name of the group to scan for geometry.

Only used when geometry_source_geometry_mode is SOURCE_GEOMETRY_GROUPS_WITH_CHILDREN or SOURCE_GEOMETRY_GROUPS_EXPLICIT.

---

float **region_merge_size** = `20.0`

- void **set_region_merge_size**(value: float)
- float **get_region_merge_size**()

Any regions with a size smaller than this will be merged with larger regions if possible.

Note: This value will be squared to calculate the number of cells. For example, a value of 20 will set the number of cells to 400.

---

float **region_min_size** = `2.0`

- void **set_region_min_size**(value: float)
- float **get_region_min_size**()

The minimum size of a region for it to be created.

Note: This value will be squared to calculate the minimum number of cells allowed to form isolated island areas. For example, a value of 8 will set the number of cells to 64.

SamplePartitionType **sample_partition_type** = `0`

- void **set_sample_partition_type**(value: SamplePartitionType)
- SamplePartitionType **get_sample_partition_type**()

Partitioning algorithm for creating the navigation mesh polys. See SamplePartitionType for possible values.

---

float **vertices_per_polygon** = `6.0`

- void **set_vertices_per_polygon**(value: float)
- float **get_vertices_per_polygon**()

The maximum number of vertices allowed for polygons generated during the contour to polygon conversion process.

Method Descriptions

void **add_polygon**(polygon: PackedInt32Array)

Adds a polygon using the indices of the vertices you get when calling get_vertices.

---

void **clear**()

Clears the internal arrays for vertices and polygon indices.

---

void **clear_polygons**()

Clears the array of polygons, but it doesn't clear the array of vertices.

---

void **create_from_mesh**(mesh: Mesh)

Initializes the navigation mesh by setting the vertices and indices according to a Mesh.

Note: The given `mesh` must be of type Mesh.PRIMITIVE_TRIANGLES and have an index array.

---

bool **get_collision_mask_value**(layer_number: int) const

Returns whether or not the specified layer of the geometry_collision_mask is enabled, given a `layer_number` between 1 and 32.

PackedInt32Array **get_polygon**(idx: int)

Returns a PackedInt32Array containing the indices of the vertices of a created polygon.

---

int **get_polygon_count**() const

Returns the number of polygons in the navigation mesh.

PackedVector3Array **get_vertices**() const

Returns a PackedVector3Array containing all the vertices being used to create the polygons.

---

void **set_collision_mask_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the geometry_collision_mask, given a `layer_number` between 1 and 32.

---

void **set_vertices**(vertices: PackedVector3Array)

Sets the vertices that can be then indexed to create polygons with the add_polygon method.

NavigationMeshSourceGeometryData2D

Experimental: This class may be changed or removed in future versions.

Inherits: Resource **<** RefCounted **<** Object

Container for parsed source geometry data used in navigation mesh baking.

Description

Container for parsed source geometry data used in navigation mesh baking.

Methods

void	add_obstruction_outline(shape_outline: PackedVector2Array)
void	add_projected_obstruction(vertices: PackedVector2Array, carve: bool)
void	add_traversable_outline(shape_outline: PackedVector2Array)
void	append_obstruction_outlines(obstruction_outlines: Array[PackedVector2Array])
void	append_traversable_outlines(traversable_outlines: Array[PackedVector2Array])
void	clear()

void	clear_projected_obstructions()
Array[PackedVector2Array]	get_obstruction_outlines() const
Array	get_projected_obstructions() const
Array[PackedVector2Array]	get_traversable_outlines() const
bool	has_data()
void	merge(other_geometry: NavigationMeshSourceGeometryData2D)
void	set_obstruction_outlines(obstruction_outlines: Array[PackedVector2Array])
void	set_projected_obstructions(projected_obstructions: Array)
void	set_traversable_outlines(traversable_outlines: Array[PackedVector2Array])

Method Descriptions

void **add_obstruction_outline**(shape_outline: PackedVector2Array) 🔗

Adds the outline points of a shape as obstructed area.

---

void **add_projected_obstruction**(vertices: PackedVector2Array, carve: bool) 🔗

Adds a projected obstruction shape to the source geometry. If `carve` is `true` the carved shape will not be affected by additional offsets (e.g. agent radius) of the navigation mesh baking process.

---

void **add_traversable_outline**(shape_outline: PackedVector2Array)

Adds the outline points of a shape as traversable area.

---

void **append_obstruction_outlines**(obstruction_outlines: Array[PackedVector2Array])

Appends another array of `obstruction_outlines` at the end of the existing obstruction outlines array.

---

void **append_traversable_outlines**(traversable_outlines: Array[PackedVector2Array])

Appends another array of `traversable_outlines` at the end of the existing traversable outlines array.

---

void **clear**()

Clears the internal data.

---

void **clear_projected_obstructions**()

Clears all projected obstructions.

Array[PackedVector2Array] **get_obstruction_outlines**() const

Returns all the obstructed area outlines arrays.

Array **get_projected_obstructions**() const

Returns the projected obstructions as an Array of dictionaries. Each Dictionary contains the following entries:

- `vertices` - A PackedFloat32Array that defines the outline points of the projected shape.
- `carve` - A bool that defines how the projected shape affects the navigation mesh baking. If `true` the projected shape will not be affected by addition offsets, e.g. agent radius.

Array[PackedVector2Array] **get_traversable_outlines**() const 🔗

Returns all the traversable area outlines arrays.

---

bool **has_data**() 🔗

Returns `true` when parsed source geometry data exists.

---

void **merge**(other_geometry: NavigationMeshSourceGeometryData2D) 🔗

Adds the geometry data of another **NavigationMeshSourceGeometryData2D** to the navigation mesh baking data.

---

void **set_obstruction_outlines**(obstruction_outlines: Array[PackedVector2Array]) 🔗

Sets all the obstructed area outlines arrays.

---

void **set_projected_obstructions**(projected_obstructions: Array) 🔗

Sets the projected obstructions with an Array of Dictionaries with the following key value pairs:

GDScript

```
"vertices" : PackedFloat32Array
"carve" : bool
```

---

void **set_traversable_outlines**(traversable_outlines: Array[PackedVector2Array]) 🔗

Sets all the traversable area outlines arrays.

NavigationMeshSourceGeometryData3D

Experimental: This class may be changed or removed in future versions.

Inherits: Resource **<** RefCounted **<** Object

Container for parsed source geometry data used in navigation mesh baking.

Description

Container for parsed source geometry data used in navigation mesh baking.

Methods

void	add_faces(faces: PackedVector3Array, xform: Transform3D)
void	add_mesh(mesh: Mesh, xform: Transform3D)
void	add_mesh_array(mesh_array: Array, xform: Transform3D)
void	add_projected_obstruction(vertices: PackedVector3Array, elevation: float, height: float, carve: bool)

void	append_arrays(vertices: PackedFloat32Array, indices: PackedInt32Array)
void	clear()
void	clear_projected_obstructions()
PackedInt32Array	get_indices() const
Array	get_projected_obstructions() const
PackedFloat32Array	get_vertices() const
bool	has_data()
void	merge(other_geometry: NavigationMeshSourceGeometryData3D)
void	set_indices(indices: PackedInt32Array)
void	set_projected_obstructions(projected_obstructions: Array)
void	set_vertices(vertices: PackedFloat32Array)

Method Descriptions

void **add_faces**(faces: PackedVector3Array, xform: Transform3D)

Adds an array of vertex positions to the geometry data for navigation mesh baking to form triangulated faces. For each face the array must have three vertex positions in clockwise winding order. Since NavigationMesh resources have no transform, all vertex positions need to be offset by the node's transform using `xform`.

---

void **add_mesh**(mesh: Mesh, xform: Transform3D)

Adds the geometry data of a Mesh resource to the navigation mesh baking data. The mesh must have valid triangulated mesh data to be considered. Since NavigationMesh resources have no transform, all vertex positions need to be offset by the node's transform using `xform`.

---

void **add_mesh_array**(mesh_array: Array, xform: Transform3D)

Adds an Array the size of Mesh.ARRAY_MAX and with vertices at index Mesh.ARRAY_VERTEX and indices at index Mesh.ARRAY_INDEX to the navigation mesh baking data. The array must have valid triangulated mesh data to be considered. Since NavigationMesh resources have no transform, all vertex positions need to be offset by the node's transform using `xform`.

---

void **add_projected_obstruction**(vertices: PackedVector3Array, elevation: float, height: float, carve: bool)

Adds a projected obstruction shape to the source geometry. The `vertices` are considered projected on a xz-axes plane, placed at the global y-axis `elevation` and extruded by `height`. If `carve` is `true` the carved shape will not be affected by additional offsets (e.g. agent radius) of the navigation mesh baking process.

---

void **append_arrays**(vertices: PackedFloat32Array, indices: PackedInt32Array)

Appends arrays of `vertices` and `indices` at the end of the existing arrays. Adds the existing index as an offset to the appended indices.

void **clear**()

Clears the internal data.

void **clear_projected_obstructions**()

Clears all projected obstructions.

PackedInt32Array **get_indices**() const

Returns the parsed source geometry data indices array.

Array **get_projected_obstructions**() const

Returns the projected obstructions as an Array of dictionaries. Each Dictionary contains the following entries:

- `vertices` - A PackedFloat32Array that defines the outline points of the projected shape.
- `elevation` - A float that defines the projected shape placement on the y-axis.
- `height` - A float that defines how much the projected shape is extruded along the y-axis.
- `carve` - A bool that defines how the obstacle affects the navigation mesh baking. If `true` the projected shape will not be affected by addition offsets, e.g. agent radius.

PackedFloat32Array **get_vertices**() const 🔗

Returns the parsed source geometry data vertices array.

---

bool **has_data**() 🔗

Returns `true` when parsed source geometry data exists.

---

void **merge**(other_geometry: NavigationMeshSourceGeometryData3D) 🔗

Adds the geometry data of another **NavigationMeshSourceGeometryData3D** to the navigation mesh baking data.

---

void **set_indices**(indices: PackedInt32Array) 🔗

Sets the parsed source geometry data indices. The indices need to be matched with appropriated vertices.

Warning: Inappropriate data can crash the baking process of the involved third-party libraries.

---

void **set_projected_obstructions**(projected_obstructions: Array) 🔗

Sets the projected obstructions with an Array of Dictionaries with the following key value pairs:

GDScript

```
"vertices" : PackedFloat32Array
"elevation" : float
"height" : float
"carve" : bool
```

---

void **set_vertices**(vertices: PackedFloat32Array)

Sets the parsed source geometry data vertices. The vertices need to be matched with appropriated indices.

Warning: Inappropriate data can crash the baking process of the involved third-party libraries.

NavigationPolygon

Experimental: This class may be changed or removed in future versions.

Inherits: Resource **<** RefCounted **<** Object

A 2D navigation mesh that describes a traversable surface for pathfinding.

Description

A navigation mesh can be created either by baking it with the help of the NavigationServer2D, or by adding vertices and convex polygon indices arrays manually.

To bake a navigation mesh at least one outline needs to be added that defines the outer bounds of the baked area.

GDScript

```
var new_navigation_mesh = NavigationPolygon.new()
var bounding_outline = PackedVector2Array([Vector2(0, 0), Vector2(0, 50), Vector2(50, 50),
Vector2(50, 0)])
new_navigation_mesh.add_outline(bounding_outline)
NavigationServer2D.bake_from_source_geometry_data(new_navigation_mesh,
NavigationMeshSourceGeometryData2D.new());
$NavigationRegion2D.navigation_polygon = new_navigation_mesh
```

C#

```
var newNavigationMesh = new NavigationPolygon();
var boundingOutline = new Vector2[] { new Vector2(0, 0), new Vector2(0, 50), new Vector2(50,
50), new Vector2(50, 0) };
newNavigationMesh.AddOutline(boundingOutline);
NavigationServer2D.BakeFromSourceGeometryData(newNavigationMesh, new
NavigationMeshSourceGeometryData2D());
GetNode<NavigationRegion2D>("NavigationRegion2D").NavigationPolygon = newNavigationMesh;
```

Adding vertices and polygon indices manually.

GDScript

```
var new_navigation_mesh = NavigationPolygon.new()
var new_vertices = PackedVector2Array([Vector2(0, 0), Vector2(0, 50), Vector2(50, 50),
Vector2(50, 0)])
new_navigation_mesh.vertices = new_vertices
var new_polygon_indices = PackedInt32Array([0, 1, 2, 3])
new_navigation_mesh.add_polygon(new_polygon_indices)
$NavigationRegion2D.navigation_polygon = new_navigation_mesh
```

C#

```
var newNavigationMesh = new NavigationPolygon();
var newVertices = new Vector2[] { new Vector2(0, 0), new Vector2(0, 50), new Vector2(50, 50),
new Vector2(50, 0) };
newNavigationMesh.Vertices = newVertices;
var newPolygonIndices = new int[] { 0, 1, 2, 3 };
```

```
newNavigationMesh.AddPolygon(newPolygonIndices);
GetNode<NavigationRegion2D>("NavigationRegion2D").NavigationPolygon = newNavigationMesh;
```

Tutorials

- Using NavigationMeshes
- Navigation Polygon 2D Demo [https://godotengine.org/asset-library/asset/2722]

Properties

float	agent_radius	`10.0`
Rect2	baking_rect	`Rect2(0, 0, 0, 0)`
Vector2	baking_rect_offset	`Vector2(0, 0)`
float	border_size	`0.0`
float	cell_size	`1.0`
int	parsed_collision_mask	`4294967295`
ParsedGeometryType	parsed_geometry_type	`2`
StringName	source_geometry_group_name	`&"navigation_polygon_source_geometry_group"`
SourceGeometryMode	source_geometry_mode	`0`

Methods

void	add_outline(outline: PackedVector2Array)
void	add_outline_at_index(outline: PackedVector2Array, index: int)
void	add_polygon(polygon: PackedInt32Array)
void	clear()

void	clear_outlines()
void	clear_polygons()
NavigationMesh	get_navigation_mesh()
PackedVector2Array	get_outline(idx: int) const
int	get_outline_count() const
bool	get_parsed_collision_mask_value(layer_number: int) const
PackedInt32Array	get_polygon(idx: int)
int	get_polygon_count() const
PackedVector2Array	get_vertices() const
void	make_polygons_from_outlines()
void	remove_outline(idx: int)
void	set_outline(idx: int, outline: PackedVector2Array)
void	set_parsed_collision_mask_value(layer_number: int, value: bool)
void	set_vertices(vertices: PackedVector2Array)

Enumerations

enum **ParsedGeometryType**: 🔗

ParsedGeometryType **PARSED_GEOMETRY_MESH_INSTANCES** = 0

Parses mesh instances as obstruction geometry. This includes Polygon2D, MeshInstance2D, MultiMeshInstance2D, and TileMap nodes.

Meshes are only parsed when they use a 2D vertices surface format.

ParsedGeometryType **PARSED_GEOMETRY_STATIC_COLLIDERS** = 1

Parses StaticBody2D and TileMap colliders as obstruction geometry. The collider should be in any of the layers specified by parsed_collision_mask.

ParsedGeometryType **PARSED_GEOMETRY_BOTH** = `2`

Both PARSED_GEOMETRY_MESH_INSTANCES and PARSED_GEOMETRY_STATIC_COLLIDERS.

ParsedGeometryType **PARSED_GEOMETRY_MAX** = `3`

Represents the size of the ParsedGeometryType enum.

---

enum **SourceGeometryMode**:

SourceGeometryMode **SOURCE_GEOMETRY_ROOT_NODE_CHILDREN** = `0`

Scans the child nodes of the root node recursively for geometry.

SourceGeometryMode **SOURCE_GEOMETRY_GROUPS_WITH_CHILDREN** = `1`

Scans nodes in a group and their child nodes recursively for geometry. The group is specified by source_geometry_group_name.

SourceGeometryMode **SOURCE_GEOMETRY_GROUPS_EXPLICIT** = `2`

Uses nodes in a group for geometry. The group is specified by source_geometry_group_name.

SourceGeometryMode **SOURCE_GEOMETRY_MAX** = `3`

Represents the size of the SourceGeometryMode enum.

Property Descriptions

float **agent_radius** = `10.0`

- void **set_agent_radius**(value: float)
- float **get_agent_radius**()

The distance to erode/shrink the walkable surface when baking the navigation mesh.

Rect2 **baking_rect** = `Rect2(0, 0, 0, 0)`

- void **set_baking_rect**(value: Rect2)
- Rect2 **get_baking_rect**()

If the baking Rect2 has an area the navigation mesh baking will be restricted to its enclosing area.

Vector2 **baking_rect_offset** = `Vector2(0, 0)`

- void **set_baking_rect_offset**(value: Vector2)
- Vector2 **get_baking_rect_offset**()

The position offset applied to the baking_rect Rect2.

float **border_size** = `0.0`

- void **set_border_size**(value: float)
- float **get_border_size**()

The size of the non-navigable border around the bake bounding area defined by the baking_rect Rect2.

In conjunction with the baking_rect the border size can be used to bake tile aligned navigation meshes without the tile edges being shrunk by agent_radius.

float **cell_size** = `1.0`

- void **set_cell_size**(value: float)
- float **get_cell_size**()

The cell size used to rasterize the navigation mesh vertices. Must match with the cell size on the navigation map.

int **parsed_collision_mask** = `4294967295`

- void **set_parsed_collision_mask**(value: int)
- int **get_parsed_collision_mask**()

The physics layers to scan for static colliders.

Only used when parsed_geometry_type is PARSED_GEOMETRY_STATIC_COLLIDERS or PARSED_GEOMETRY_BOTH.

ParsedGeometryType **parsed_geometry_type** = `2`

- void **set_parsed_geometry_type**(value: ParsedGeometryType)
- ParsedGeometryType **get_parsed_geometry_type**()

Determines which type of nodes will be parsed as geometry. See ParsedGeometryType for possible values.

StringName **source_geometry_group_name** = `&"navigation_polygon_source_geometry_group"`

- void **set_source_geometry_group_name**(value: StringName)
- StringName **get_source_geometry_group_name**()

The group name of nodes that should be parsed for baking source geometry.

Only used when source_geometry_mode is SOURCE_GEOMETRY_GROUPS_WITH_CHILDREN or SOURCE_GEOMETRY_GROUPS_EXPLICIT.

SourceGeometryMode **source_geometry_mode** = `0`

- void **set_source_geometry_mode**(value: SourceGeometryMode)
- SourceGeometryMode **get_source_geometry_mode**()

The source of the geometry used when baking. See SourceGeometryMode for possible values.

Method Descriptions

void **add_outline**(outline: PackedVector2Array)

Appends a PackedVector2Array that contains the vertices of an outline to the internal array that contains all the outlines.

void **add_outline_at_index**(outline: PackedVector2Array, index: int)

Adds a PackedVector2Array that contains the vertices of an outline to the internal array that contains all the outlines at a fixed position.

void **add_polygon**(polygon: PackedInt32Array)

Adds a polygon using the indices of the vertices you get when calling get_vertices.

void **clear**()

Clears the internal arrays for vertices and polygon indices.

void **clear_outlines**()

Clears the array of the outlines, but it doesn't clear the vertices and the polygons that were created by them.

void **clear_polygons**()

Clears the array of polygons, but it doesn't clear the array of outlines and vertices.

NavigationMesh **get_navigation_mesh**()

Returns the NavigationMesh resulting from this navigation polygon. This navigation mesh can be used to update the navigation mesh of a region with the NavigationServer3D.region_set_navigation_mesh API directly (as 2D uses the 3D server behind the scene).

PackedVector2Array **get_outline**(idx: int) const

Returns a PackedVector2Array containing the vertices of an outline that was created in the editor or by script.

int **get_outline_count**() const

Returns the number of outlines that were created in the editor or by script.

bool **get_parsed_collision_mask_value**(layer_number: int) const

Returns whether or not the specified layer of the parsed_collision_mask is enabled, given a `layer_number` between 1 and 32.

PackedInt32Array **get_polygon**(idx: int)

Returns a PackedInt32Array containing the indices of the vertices of a created polygon.

int **get_polygon_count**() const

Returns the count of all polygons.

PackedVector2Array **get_vertices**() const

Returns a PackedVector2Array containing all the vertices being used to create the polygons.

void **make_polygons_from_outlines**()

Deprecated: Use NavigationServer2D.parse_source_geometry_data and NavigationServer2D.bake_from_source_geometry_data instead.

Creates polygons from the outlines added in the editor or by script.

void **remove_outline**(idx: int)

Removes an outline created in the editor or by script. You have to call make_polygons_from_outlines for the polygons to update.

void **set_outline**(idx: int, outline: PackedVector2Array)

Changes an outline created in the editor or by script. You have to call make_polygons_from_outlines for the polygons to update.

void **set_parsed_collision_mask_value**(layer_number: int, value: bool)

Based on `value`, enables or disables the specified layer in the parsed_collision_mask, given a `layer_number` between 1 and 32.

void **set_vertices**(vertices: PackedVector2Array)

Sets the vertices that can be then indexed to create polygons with the add_polygon method.

Noise

Inherits: Resource **<** RefCounted **<** Object

Inherited By: FastNoiseLite

Abstract base class for noise generators.

Description

This class defines the interface for noise generation libraries to inherit from.

A default get_seamless_image implementation is provided for libraries that do not provide seamless noise. This function requests a larger image from the get_image method, reverses the quadrants of the image, then uses the strips of extra width to blend over the seams.

Inheriting noise classes can optionally override this function to provide a more optimal algorithm.

Methods

Image	get_image(width: int, height: int, invert: bool = false, in_3d_space: bool = false, normalize: bool = true) const
Array[Image]	get_image_3d(width: int, height: int, depth: int, invert: bool = false, normalize: bool = true) const

float	get_noise_1d(x: float) const
float	get_noise_2d(x: float, y: float) const
float	get_noise_2dv(v: Vector2) const
float	get_noise_3d(x: float, y: float, z: float) const
float	get_noise_3dv(v: Vector3) const
Image	get_seamless_image(width: int, height: int, invert: bool = false, in_3d_space: bool = false, skirt: float = 0.1, normalize: bool = true) const
Array[Image]	get_seamless_image_3d(width: int, height: int, depth: int, invert: bool = false, skirt: float = 0.1, normalize: bool = true) const

Method Descriptions

Image **get_image**(width: int, height: int, invert: bool = false, in_3d_space: bool = false, normalize: bool = true) const 🔗

Returns an Image containing 2D noise values.

Note: With `normalize` set to `false`, the default implementation expects the noise generator to return values in the range `-1.0` to

`1.0`.

Array[Image] **get_image_3d**(width: int, height: int, depth: int, invert: bool = false, normalize: bool = true) const

Returns an Array of Images containing 3D noise values for use with ImageTexture3D.create.

Note: With `normalize` set to `false`, the default implementation expects the noise generator to return values in the range `-1.0` to `1.0`.

---

float **get_noise_1d**(x: float) const

Returns the 1D noise value at the given (x) coordinate.

---

float **get_noise_2d**(x: float, y: float) const

Returns the 2D noise value at the given position.

---

float **get_noise_2dv**(v: Vector2) const

Returns the 2D noise value at the given position.

---

float **get_noise_3d**(x: float, y: float, z: float) const

Returns the 3D noise value at the given position.

---

float **get_noise_3dv**(v: Vector3) const

Returns the 3D noise value at the given position.

Image **get_seamless_image**(width: int, height: int, invert: bool = false, in_3d_space: bool = false, skirt: float = 0.1, normalize: bool = true) const 🔗

Returns an Image containing seamless 2D noise values.

Note: With `normalize` set to `false`, the default implementation expects the noise generator to return values in the range `-1.0` to `1.0`.

Array[Image] **get_seamless_image_3d**(width: int, height: int, depth: int, invert: bool = false, skirt: float = 0.1, normalize: bool = true) const 🔗

Returns an Array of Images containing seamless 3D noise values for use with ImageTexture3D.create.

Note: With `normalize` set to `false`, the default implementation expects the noise generator to return values in the range `-1.0` to `1.0`.

NoiseTexture2D

Inherits: Texture2D **<** Texture **<** Resource **<** RefCounted **<** Object

A 2D texture filled with noise generated by a Noise object.

Description

Uses the FastNoiseLite library or other noise generators to fill the texture data of your desired size. **NoiseTexture2D** can also generate normal map textures.

The class uses Threads to generate the texture data internally, so Texture2D.get_image may return `null` if the generation process has not completed yet. In that case, you need to wait for the texture to be generated before accessing the image and the generated byte data:

```
var texture = NoiseTexture2D.new()
texture.noise = FastNoiseLite.new()
await texture.changed
var image = texture.get_image()
var data = image.get_data()
```

Properties

bool	as_normal_map	`false`
float	bump_strength	`8.0`

Gradient	color_ramp	
bool	generate_mipmaps	`true`
int	height	`512`
bool	in_3d_space	`false`
bool	invert	`false`
Noise	noise	
bool	normalize	`true`
bool	resource_local_to_scene	`false` (overrides Resource)
bool	seamless	`false`
float	seamless_blend_skirt	`0.1`
int	width	`512`

Property Descriptions

bool **as_normal_map** = `false` 🔗

- void **set_as_normal_map**(value: bool)
- bool **is_normal_map**()

If `true`, the resulting texture contains a normal map created from the original noise interpreted as a bump map.

---

float **bump_strength** = `8.0` 🔗

- void **set_bump_strength**(value: float)
- float **get_bump_strength**()

Strength of the bump maps used in this texture. A higher value will make the bump maps appear larger while a lower value will make them appear softer.

Gradient **color_ramp** 🔗

- void **set_color_ramp**(value: Gradient)
- Gradient **get_color_ramp**()

A Gradient which is used to map the luminance of each pixel to a color value.

---

bool **generate_mipmaps** = `true` 🔗

- void **set_generate_mipmaps**(value: bool)
- bool **is_generating_mipmaps**()

Determines whether mipmaps are generated for this texture. Enabling this results in less texture aliasing in the distance, at the cost of increasing memory usage by roughly 33% and making the noise texture generation take longer.

Note: generate_mipmaps requires mipmap filtering to be enabled on the material using the **NoiseTexture2D** to have an effect.

---

int **height** = `512` 🔗

- void **set_height**(value: int)
- int **get_height**()

Height of the generated texture (in pixels).

---

bool **in_3d_space** = `false` 🔗

- void **set_in_3d_space**(value: bool)
- bool **is_in_3d_space**()

Determines whether the noise image is calculated in 3D space. May result in reduced contrast.

---

bool **invert** = `false` 🔗

- void **set_invert**(value: bool)
- bool **get_invert**()

If `true`, inverts the noise texture. White becomes black, black becomes white.

Noise **noise** 🔗

- void **set_noise**(value: Noise)
- Noise **get_noise**()

The instance of the Noise object.

---

bool **normalize** = `true` 🔗

- void **set_normalize**(value: bool)
- bool **is_normalized**()

If `true`, the noise image coming from the noise generator is normalized to the range `0.0` to `1.0`.

Turning normalization off can affect the contrast and allows you to generate non repeating tileable noise textures.

---

bool **seamless** = `false` 🔗

- void **set_seamless**(value: bool)
- bool **get_seamless**()

If `true`, a seamless texture is requested from the Noise resource.

Note: Seamless noise textures may take longer to generate and/or can have a lower contrast compared to non-seamless noise depending on the used Noise resource. This is because some implementations use higher dimensions for generating seamless noise.

Note: The default FastNoiseLite implementation uses the fallback path for seamless generation. If using a width or height lower than the default, you may need to increase seamless_blend_skirt to make seamless blending more effective.

---

float **seamless_blend_skirt** = `0.1` 🔗

- void **set_seamless_blend_skirt**(value: float)
- float **get_seamless_blend_skirt**()

Used for the default/fallback implementation of the seamless texture generation. It determines the distance over which the seams are blended. High values may result in less details and contrast. See Noise for further details.

Note: If using a width or height lower than the default, you may need to increase seamless_blend_skirt to make seamless blending more effective.

---

int **width** = `512`

- void **set_width**(value: int)
- int **get_width**()

Width of the generated texture (in pixels).

NoiseTexture3D

Inherits: Texture3D **<** Texture **<** Resource **<** RefCounted **<** Object

A 3D texture filled with noise generated by a Noise object.

Description

Uses the FastNoiseLite library or other noise generators to fill the texture data of your desired size.

The class uses Threads to generate the texture data internally, so Texture3D.get_data may return `null` if the generation process has not completed yet. In that case, you need to wait for the texture to be generated before accessing the image:

```
var texture = NoiseTexture3D.new()
texture.noise = FastNoiseLite.new()
await texture.changed
var data = texture.get_data()
```

Properties

Gradient	color_ramp	
int	depth	`64`
int	height	`64`

bool	invert	`false`
Noise	noise	
bool	normalize	`true`
bool	seamless	`false`
float	seamless_blend_skirt	`0.1`
int	width	`64`

Property Descriptions

Gradient **color_ramp**

- void **set_color_ramp**(value: Gradient)
- Gradient **get_color_ramp**()

A Gradient which is used to map the luminance of each pixel to a color value.

---

int **depth** = `64`

- void **set_depth**(value: int)
- int **get_depth**()

Depth of the generated texture (in pixels).

---

int **height** = `64` 🔗

- void **set_height**(value: int)
- int **get_height**()

Height of the generated texture (in pixels).

---

bool **invert** = `false` 🔗

- void **set_invert**(value: bool)
- bool **get_invert**()

If `true`, inverts the noise texture. White becomes black, black becomes white.

Noise **noise** 🔗

- void **set_noise**(value: Noise)
- Noise **get_noise**()

The instance of the Noise object.

---

bool **normalize** = `true` 🔗

- void **set_normalize**(value: bool)
- bool **is_normalized**()

If `true`, the noise image coming from the noise generator is normalized to the range `0.0` to `1.0`.

Turning normalization off can affect the contrast and allows you to generate non repeating tileable noise textures.

---

bool **seamless** = `false` 🔗

- void **set_seamless**(value: bool)
- bool **get_seamless**()

If `true`, a seamless texture is requested from the Noise resource.

Note: Seamless noise textures may take longer to generate and/or can have a lower contrast compared to non-seamless noise depending on the used Noise resource. This is because some implementations use higher dimensions for generating seamless noise.

Note: The default FastNoiseLite implementation uses the fallback path for seamless generation. If using a width, height or depth lower than the default, you may need to increase seamless_blend_skirt to make seamless blending more effective.

---

float **seamless_blend_skirt** = `0.1` 🔗

- void **set_seamless_blend_skirt**(value: float)
- float **get_seamless_blend_skirt**()

Used for the default/fallback implementation of the seamless texture generation. It determines the distance over which the seams are blended. High values may result in less details and contrast. See Noise for further details.

Note: If using a width, height or depth lower than the default, you may need to increase seamless_blend_skirt to make seamless blending more effective.

---

int **width** = `64` 🔗

- void **set_width**(value: int)

- int **get_width**()

Width of the generated texture (in pixels).

Occluder3D

Inherits: Resource **<** RefCounted **<** Object

Inherited By: ArrayOccluder3D, BoxOccluder3D, PolygonOccluder3D, QuadOccluder3D, SphereOccluder3D

Occluder shape resource for use with occlusion culling in OccluderInstance3D.

Description

Occluder3D stores an occluder shape that can be used by the engine's occlusion culling system.

See OccluderInstance3D's documentation for instructions on setting up occlusion culling.

Tutorials

- Occlusion culling

Methods

PackedInt32Array	get_indices() const
PackedVector3Array	get_vertices() const

Method Descriptions

PackedInt32Array **get_indices**() const

Returns the occluder shape's vertex indices.

PackedVector3Array **get_vertices**() const

Returns the occluder shape's vertex positions.

OccluderPolygon2D

Inherits: Resource **<** RefCounted **<** Object

Defines a 2D polygon for LightOccluder2D.

Description

Editor facility that helps you draw a 2D polygon used as resource for LightOccluder2D.

Properties

bool	closed	`true`
CullMode	cull_mode	`0`
PackedVector2Array	polygon	`PackedVector2Array()`

Enumerations

enum **CullMode**: 🔗

CullMode **CULL_DISABLED** = `0`

Culling is disabled. See cull_mode.

CullMode **CULL_CLOCKWISE** = `1`

Culling is performed in the clockwise direction. See cull_mode.

CullMode **CULL_COUNTER_CLOCKWISE** = `2`

Culling is performed in the counterclockwise direction. See cull_mode.

Property Descriptions

bool **closed** = `true` 🔗

- void **set_closed**(value: bool)
- bool **is_closed**()

If `true`, closes the polygon. A closed OccluderPolygon2D occludes the light coming from any direction. An opened OccluderPolygon2D occludes the light only at its outline's direction.

CullMode **cull_mode** = `0` 🔗

- void **set_cull_mode**(value: CullMode)
- CullMode **get_cull_mode**()

The culling mode to use.

PackedVector2Array **polygon** = `PackedVector2Array()` 🔗

- void **set_polygon**(value: PackedVector2Array)
- PackedVector2Array **get_polygon**()

A Vector2 array with the index for polygon's vertices positions.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector2Array for more details.

OggPacketSequence

Inherits: Resource **<** RefCounted **<** Object

A sequence of Ogg packets.

Description

A sequence of Ogg packets.

Properties

PackedInt64Array	granule_positions	`PackedInt64Array()`
Array[Array]	packet_data	`[]`
float	sampling_rate	`0.0`

Methods

float	get_length() const

Property Descriptions

PackedInt64Array **granule_positions** = `PackedInt64Array()`

- void **set_packet_granule_positions**(value: PackedInt64Array)
- PackedInt64Array **get_packet_granule_positions**()

Contains the granule positions for each page in this packet sequence.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedInt64Array for more details.

Array[Array] **packet_data** = `[]`

- void **set_packet_data**(value: Array[Array])
- Array[Array] **get_packet_data**()

Contains the raw packets that make up this OggPacketSequence.

---

float **sampling_rate** = `0.0`

- void **set_sampling_rate**(value: float)
- float **get_sampling_rate**()

Holds sample rate information about this sequence. Must be set by another class that actually understands the codec.

Method Descriptions

float **get_length**() const

The length of this stream, in seconds.

OpenXRAction

Inherits: Resource **<** RefCounted **<** Object

An OpenXR action.

Description

This resource defines an OpenXR action. Actions can be used both for inputs (buttons, joysticks, triggers, etc.) and outputs (haptics).

OpenXR performs automatic conversion between action type and input type whenever possible. An analog trigger bound to a boolean action will thus return `false` if the trigger is depressed and `true` if pressed fully.

Actions are not directly bound to specific devices, instead OpenXR recognizes a limited number of top level paths that identify devices by usage. We can restrict which devices an action can be bound to by these top level paths. For instance an action that should only be used for hand held controllers can have the top level paths "/user/hand/left" and "/user/hand/right" associated with them. See the reserved path section in the OpenXR specification [https://www.khronos.org/registry/OpenXR/specs/1.0/html/xrspec.html#semantic-path-reserved] for more info on the top level paths.

Note that the name of the resource is used to register the action with.

Properties

ActionType	action_type	`1`
String	localized_name	`""`
PackedStringArray	toplevel_paths	`PackedStringArray()`

Enumerations

enum **ActionType**: 🔗

ActionType **OPENXR_ACTION_BOOL** = `0`

This action provides a boolean value.

ActionType **OPENXR_ACTION_FLOAT** = `1`

This action provides a float value between `0.0` and `1.0` for any analog input such as triggers.

ActionType **OPENXR_ACTION_VECTOR2** = `2`

This action provides a Vector2 value and can be bound to embedded trackpads and joysticks.

ActionType **OPENXR_ACTION_POSE** = `3`

There is currently no description for this enum. Please help us by contributing one!

Property Descriptions

ActionType **action_type** = `1` 🔗

- void **set_action_type**(value: ActionType)
- ActionType **get_action_type**()

The type of action.

String **localized_name** = `""` 🔗

- void **set_localized_name**(value: String)
- String **get_localized_name**()

The localized description of this action.

PackedStringArray **toplevel_paths** = `PackedStringArray()` 🔗

- void **set_toplevel_paths**(value: PackedStringArray)
- PackedStringArray **get_toplevel_paths**()

A collections of toplevel paths to which this action can be bound.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedStringArray for more details.

OpenXRActionMap

Inherits: Resource **<** RefCounted **<** Object

Collection of OpenXRActionSet and OpenXRInteractionProfile resources for the OpenXR module.

Description

OpenXR uses an action system similar to Godots Input map system to bind inputs and outputs on various types of XR controllers to named actions. OpenXR specifies more detail on these inputs and outputs than Godot supports.

Another important distinction is that OpenXR offers no control over these bindings. The bindings we register are suggestions, it is up to the XR runtime to offer users the ability to change these bindings. This allows the XR runtime to fill in the gaps if new hardware becomes available.

The action map therefore needs to be loaded at startup and can't be changed afterwards. This resource is a container for the entire action map.

Properties

Array	action_sets	`[ ]`
Array	interaction_profiles	`[ ]`

Methods

void	add_action_set(action_set: OpenXRActionSet)
void	add_interaction_profile(interaction_profile: OpenXRInteractionProfile)
void	create_default_action_sets()
OpenXRActionSet	find_action_set(name: String) const
OpenXRInteractionProfile	find_interaction_profile(name: String) const
OpenXRActionSet	get_action_set(idx: int) const
int	get_action_set_count() const
OpenXRInteractionProfile	get_interaction_profile(idx: int) const
int	get_interaction_profile_count() const
void	remove_action_set(action_set: OpenXRActionSet)
void	remove_interaction_profile(interaction_profile: OpenXRInteractionProfile)

Property Descriptions

Array **action_sets** = `[]` 🔗

- void **set_action_sets**(value: Array)
- Array **get_action_sets**()

Collection of OpenXRActionSets that are part of this action map.

Array **interaction_profiles** = `[]` 🔗

- void **set_interaction_profiles**(value: Array)
- Array **get_interaction_profiles**()

Collection of OpenXRInteractionProfiles that are part of this action map.

Method Descriptions

void **add_action_set**(action_set: OpenXRActionSet) 🔗

Add an action set.

---

void **add_interaction_profile**(interaction_profile: OpenXRInteractionProfile) 🔗

Add an interaction profile.

---

void **create_default_action_sets**() 🔗

Setup this action set with our default actions.

OpenXRActionSet **find_action_set**(name: String) const 🔗

Retrieve an action set by name.

OpenXRInteractionProfile **find_interaction_profile**(name: String) const

Find an interaction profile by its name (path).

OpenXRActionSet **get_action_set**(idx: int) const

Retrieve the action set at this index.

---

int **get_action_set_count**() const

Retrieve the number of actions sets in our action map.

OpenXRInteractionProfile **get_interaction_profile**(idx: int) const

Get the interaction profile at this index.

---

int **get_interaction_profile_count**() const

Retrieve the number of interaction profiles in our action map.

---

void **remove_action_set**(action_set: OpenXRActionSet)

Remove an action set.

---

void **remove_interaction_profile**(interaction_profile: OpenXRInteractionProfile)

Remove an interaction profile.

OpenXRActionSet

Inherits: Resource **<** RefCounted **<** Object

Collection of OpenXRAction resources that make up an action set.

Description

Action sets in OpenXR define a collection of actions that can be activated in unison. This allows games to easily change between different states that require different inputs or need to reinterpret inputs. For instance we could have an action set that is active when a menu is open, an action set that is active when the player is freely walking around and an action set that is active when the player is controlling a vehicle.

Action sets can contain the same action with the same name, if such action sets are active at the same time the action set with the highest priority defines which binding is active.

Properties

Array	actions	`[]`
String	localized_name	`""`
int	priority	`0`

Methods

void	add_action(action: OpenXRAction)
int	get_action_count() const
void	remove_action(action: OpenXRAction)

Property Descriptions

Array **actions** = `[]`

- void **set_actions**(value: Array)
- Array **get_actions**()

Collection of actions for this action set.

String **localized_name** = `""`

- void **set_localized_name**(value: String)
- String **get_localized_name**()

The localized name of this action set.

---

int **priority** = `0`

- void **set_priority**(value: int)
- int **get_priority**()

The priority for this action set.

Method Descriptions

void **add_action**(action: OpenXRAction)

Add an action to this action set.

---

int **get_action_count**() const

Retrieve the number of actions in our action set.

---

void **remove_action**(action: OpenXRAction)

Remove an action from this action set.

OpenXRInteractionProfile

Inherits: Resource **<** RefCounted **<** Object

Suggested bindings object for OpenXR.

Description

This object stores suggested bindings for an interaction profile. Interaction profiles define the metadata for a tracked XR device such as an XR controller.

For more information see the interaction profiles info in the OpenXR specification [https://www.khronos.org/registry/OpenXR/specs/1.0/html/xrspec.html#semantic-path-interaction-profiles].

Properties

Array	bindings	[]
String	interaction_profile_path	""

Methods

OpenXRIPBinding	get_binding(index: int) const

int	get_binding_count() const

Property Descriptions

Array **bindings** = `[]`

- void **set_bindings**(value: Array)
- Array **get_bindings**()

Action bindings for this interaction profile.

String **interaction_profile_path** = `""`

- void **set_interaction_profile_path**(value: String)
- String **get_interaction_profile_path**()

The interaction profile path identifying the XR device.

Method Descriptions

OpenXRIPBinding **get_binding**(index: int) const

Retrieve the binding at this index.

---

int **get_binding_count**() const

Get the number of bindings in this interaction profile.

OpenXRIPBinding

Inherits: Resource **<** RefCounted **<** Object

Defines a binding between an OpenXRAction and an XR input or output.

Description

This binding resource binds an OpenXRAction to inputs or outputs. As most controllers have left hand and right versions that are handled by the same interaction profile we can specify multiple bindings. For instance an action "Fire" could be bound to both "/user/hand/left/input/trigger" and "/user/hand/right/input/trigger".

Properties

OpenXRAction	action	
PackedStringArray	paths	`PackedStringArray()`

Methods

void	add_path(path: String)
int	get_path_count() const

bool	has_path(path: String) const
void	remove_path(path: String)

Property Descriptions

OpenXRAction **action**

- void **set_action**(value: OpenXRAction)
- OpenXRAction **get_action**()

OpenXRAction that is bound to these paths.

PackedStringArray **paths** = `PackedStringArray()`

- void **set_paths**(value: PackedStringArray)
- PackedStringArray **get_paths**()

Paths that define the inputs or outputs bound on the device.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedStringArray for more details.

Method Descriptions

void **add_path**(path: String)

Add an input/output path to this binding.

---

int **get_path_count**() const

Get the number of input/output paths in this binding.

---

bool **has_path**(path: String) const

Returns `true` if this input/output path is part of this binding.

---

void **remove_path**(path: String)

Removes this input/output path from this binding.

OptimizedTranslation

Inherits: Translation **<** Resource **<** RefCounted **<** Object

An optimized translation, used by default for CSV Translations.

Description

An optimized translation, used by default for CSV Translations. Uses real-time compressed translations, which results in very small dictionaries.

Methods

void	generate(from: Translation)

Method Descriptions

void **generate**(from: Translation) 🔗

Generates and sets an optimized translation from the given Translation resource.

ORMMaterial3D

Inherits: BaseMaterial3D **<** Material **<** Resource **<** RefCounted **<** Object

A PBR (Physically Based Rendering) material to be used on 3D objects. Uses an ORM texture.

Description

ORMMaterial3D's properties are inherited from BaseMaterial3D. Unlike StandardMaterial3D, ORMMaterial3D uses a single texture for ambient occlusion, roughness and metallic maps, known as an ORM texture.

Tutorials

- Standard Material 3D and ORM Material 3D

PackedDataContainer

Inherits: Resource < RefCounted < Object

Efficiently packs and serializes Array or Dictionary.

Description

PackedDataContainer can be used to efficiently store data from untyped containers. The data is packed into raw bytes and can be saved to file. Only Array and Dictionary can be stored this way.

You can retrieve the data by iterating on the container, which will work as if iterating on the packed data itself. If the packed container is a Dictionary, the data can be retrieved by key names (String/StringName only).

```
var data = { "key": "value", "another_key": 123, "lock":
Vector2() }
var packed = PackedDataContainer.new()
packed.pack(data)
ResourceSaver.save(packed, "packed_data.res")
```

```
var container = load("packed_data.res")
for key in container:
    prints(key, container[key])

# Prints:
# key value
# lock (0, 0)
# another_key 123
```

Nested containers will be packed recursively. While iterating, they will be returned as PackedDataContainerRef.

Methods

Error	pack(value: Variant)
int	size() const

Method Descriptions

Error **pack**(value: Variant)

Packs the given container into a binary representation. The `value` must be either Array or Dictionary, any other type will result in invalid data error.

Note: Subsequent calls to this method will overwrite the existing data.

int **size**() const

Returns the size of the packed container (see Array.size and Dictionary.size).

PackedScene

Inherits: Resource **<** RefCounted **<** Object

An abstraction of a serialized scene.

Description

A simplified interface to a scene file. Provides access to operations and checks that can be performed on the scene resource itself.

Can be used to save a node to a file. When saving, the node as well as all the nodes it owns get saved (see Node.owner property).

Note: The node doesn't need to own itself.

Example of loading a saved scene:

GDScript

```
# Use load() instead of preload() if the path isn't known at
compile-time.
var scene = preload("res://scene.tscn").instantiate()
# Add the node as a child of the node the script is attached
to.
add_child(scene)
```

C#

```
// C# has no preload, so you have to always use
ResourceLoader.Load<PackedScene>().
var scene = ResourceLoader.Load<PackedScene>
("res://scene.tscn").Instantiate();
// Add the node as a child of the node the script is attached
```

```
to.
AddChild(scene);
```

Example of saving a node with different owners: The following example creates 3 objects: Node2D (`node`), RigidBody2D (`body`) and CollisionObject2D (`collision`). `collision` is a child of `body` which is a child of `node`. Only `body` is owned by `node` and pack will therefore only save those two nodes, but not `collision`.

GDScript

```
# Create the objects.
var node = Node2D.new()
var body = RigidBody2D.new()
var collision = CollisionShape2D.new()

# Create the object hierarchy.
body.add_child(collision)
node.add_child(body)

# Change owner of `body`, but not of `collision`.
body.owner = node
var scene = PackedScene.new()

# Only `node` and `body` are now packed.
var result = scene.pack(node)
if result == OK:
    var error = ResourceSaver.save(scene,
"res://path/name.tscn")  # Or "user://..."
    if error != OK:
        push_error("An error occurred while saving the scene to
disk.")
```

C#

```
// Create the objects.
var node = new Node2D();
var body = new RigidBody2D();
var collision = new CollisionShape2D();
```

```
// Create the object hierarchy.
body.AddChild(collision);
node.AddChild(body);

// Change owner of `body`, but not of `collision`.
body.Owner = node;
var scene = new PackedScene();

// Only `node` and `body` are now packed.
Error result = scene.Pack(node);
if (result == Error.Ok)
{
    Error error = ResourceSaver.Save(scene, "res://path/name.tscn"); // Or "user://..."
    if (error != Error.Ok)
    {
        GD.PushError("An error occurred while saving the scene to disk.");
    }
}
```

Tutorials

- [2D Role Playing Game (RPG) Demo](https://godotengine.org/asset-library/asset/2729) [https://godotengine.org/asset-library/asset/2729]

Properties

Dictionary	_bundled	`{ "conn_count": 0, "conns": PackedInt32Array(), "editable_instances": [], "names": PackedStringArray(), "node_count": 0, "node_paths": [], "nodes": PackedInt32Array(), "variants": [], "version": 3 }`

Methods

bool	can_instantiate() const
SceneState	get_state() const
Node	instantiate(edit_state: GenEditState = 0) const
Error	pack(path: Node)

Enumerations

enum **GenEditState**: 🔗

GenEditState **GEN_EDIT_STATE_DISABLED** = `0`

If passed to instantiate, blocks edits to the scene state.

GenEditState **GEN_EDIT_STATE_INSTANCE** = `1`

If passed to instantiate, provides local scene resources to the local scene.

Note: Only available in editor builds.

GenEditState **GEN_EDIT_STATE_MAIN** = `2`

If passed to instantiate, provides local scene resources to the local scene. Only the main scene should receive the main edit state.

Note: Only available in editor builds.

GenEditState **GEN_EDIT_STATE_MAIN_INHERITED** = `3`

It's similar to GEN_EDIT_STATE_MAIN, but for the case where the scene is being instantiated to be the base of another one.

Note: Only available in editor builds.

Property Descriptions

Dictionary **_bundled** = `{ "conn_count": 0, "conns": PackedInt32Array(), "editable_instances": [], "names": PackedStringArray(), "node_count": 0, "node_paths": [], "nodes": PackedInt32Array(), "variants": [], "version": 3 }` 🔗

A dictionary representation of the scene contents.

Available keys include "names" and "variants" for resources, "node_count", "nodes", "node_paths" for nodes, "editable_instances" for paths to overridden nodes, "conn_count" and "conns" for signal connections, and "version" for the format style of the PackedScene.

Method Descriptions

bool **can_instantiate**() const 🔗

Returns `true` if the scene file has nodes.

SceneState **get_state**() const 🔗

Returns the SceneState representing the scene file contents.

Node **instantiate**(edit_state: GenEditState = 0) const 🔗

Instantiates the scene's node hierarchy. Triggers child scene instantiation(s). Triggers a Node.NOTIFICATION_SCENE_INSTANTIATED notification on the root node.

Error **pack**(path: Node) 🔗

Packs the `path` node, and all owned sub-nodes, into this **PackedScene**. Any existing data will be cleared. See Node.owner.

PanoramaSkyMaterial

Inherits: Material < Resource < RefCounted < Object

A material that provides a special texture to a Sky, usually an HDR panorama.

Description

A resource referenced in a Sky that is used to draw a background. **PanoramaSkyMaterial** functions similar to skyboxes in other engines, except it uses an equirectangular sky map instead of a Cubemap.

Using an HDR panorama is strongly recommended for accurate, high-quality reflections. Godot supports the Radiance HDR (`.hdr`) and OpenEXR (`.exr`) image formats for this purpose.

You can use this tool [https://danilw.github.io/GLSL-howto/cubemap_to_panorama_js/cubemap_to_panorama.html] to convert a cubemap to an equirectangular sky map.

Properties

float	energy_multiplier	`1.0`
bool	filter	`true`
Texture2D	panorama	

Property Descriptions

float **energy_multiplier** = `1.0`

- void **set_energy_multiplier**(value: float)
- float **get_energy_multiplier**()

The sky's overall brightness multiplier. Higher values result in a brighter sky.

---

bool **filter** = `true`

- void **set_filtering_enabled**(value: bool)
- bool **is_filtering_enabled**()

A boolean value to determine if the background texture should be filtered or not.

Texture2D **panorama**

- void **set_panorama**(value: Texture2D)
- Texture2D **get_panorama**()

Texture2D to be applied to the **PanoramaSkyMaterial**.

ParticleProcessMaterial

Inherits: Material **<** Resource **<** RefCounted **<** Object

Holds a particle configuration for GPUParticles2D or GPUParticles3D nodes.

Description

ParticleProcessMaterial defines particle properties and behavior. It is used in the `process_material` of the GPUParticles2D and GPUParticles3D nodes. Some of this material's properties are applied to each particle when emitted, while others can have a CurveTexture or a GradientTexture1D applied to vary numerical or color values over the lifetime of the particle.

Properties

Texture2D	alpha_curve	
Texture2D	angle_curve	
float	angle_max	`0.0`
float	angle_min	`0.0`
Texture2D	angular_velocity_curve	

float	angular_velocity_max	0.0
float	angular_velocity_min	0.0
Texture2D	anim_offset_curve	
float	anim_offset_max	0.0
float	anim_offset_min	0.0
Texture2D	anim_speed_curve	
float	anim_speed_max	0.0
float	anim_speed_min	0.0
bool	attractor_interaction_enabled	true
float	collision_bounce	
float	collision_friction	
CollisionMode	collision_mode	0
bool	collision_use_scale	false

Color	color	`Color(1, 1, 1, 1)`
Texture2D	color_initial_ramp	
Texture2D	color_ramp	
Texture2D	damping_curve	
float	damping_max	`0.0`
float	damping_min	`0.0`
Vector3	direction	`Vector3(1, 0, 0)`
Texture2D	directional_velocity_curve	
float	directional_velocity_max	
float	directional_velocity_min	
Vector3	emission_box_extents	
Texture2D	emission_color_texture	

Texture2D	emission_curve	
Texture2D	emission_normal_texture	
int	emission_point_count	
Texture2D	emission_point_texture	
Vector3	emission_ring_axis	
float	emission_ring_height	
float	emission_ring_inner_radius	
float	emission_ring_radius	
EmissionShape	emission_shape	`0`
Vector3	emission_shape_offset	`Vector3(0, 0, 0)`
Vector3	emission_shape_scale	`Vector3(1, 1, 1)`
float	emission_sphere_radius	

float	flatness	`0.0`
Vector3	gravity	`Vector3(0, -9.8, 0)`
Texture2D	hue_variation_curve	
float	hue_variation_max	`0.0`
float	hue_variation_min	`0.0`
float	inherit_velocity_ratio	`0.0`
float	initial_velocity_max	`0.0`
float	initial_velocity_min	`0.0`
float	lifetime_randomness	`0.0`
Texture2D	linear_accel_curve	
float	linear_accel_max	`0.0`
float	linear_accel_min	`0.0`

Texture2D	orbit_velocity_curve	
float	orbit_velocity_max	`0.0`
float	orbit_velocity_min	`0.0`
bool	particle_flag_align_y	`false`
bool	particle_flag_damping_as_friction	`false`
bool	particle_flag_disable_z	`false`
bool	particle_flag_rotate_y	`false`
Texture2D	radial_accel_curve	
float	radial_accel_max	`0.0`
float	radial_accel_min	`0.0`
Texture2D	radial_velocity_curve	
float	radial_velocity_max	`0.0`
float	radial_velocity_min	`0.0`

Texture2D	scale_curve	
float	scale_max	`1.0`
float	scale_min	`1.0`
Texture2D	scale_over_velocity_curve	
float	scale_over_velocity_max	`0.0`
float	scale_over_velocity_min	`0.0`
float	spread	`45.0`
int	sub_emitter_amount_at_collision	
int	sub_emitter_amount_at_end	
float	sub_emitter_frequency	
bool	sub_emitter_keep_velocity	`false`
SubEmitterMode	sub_emitter_mode	`0`

Texture2D	tangential_accel_curve	
float	tangential_accel_max	`0.0`
float	tangential_accel_min	`0.0`
bool	turbulence_enabled	`false`
float	turbulence_influence_max	`0.1`
float	turbulence_influence_min	`0.1`
Texture2D	turbulence_influence_over_life	
float	turbulence_initial_displacement_max	`0.0`
float	turbulence_initial_displacement_min	`0.0`
float	turbulence_noise_scale	`9.0`
Vector3	turbulence_noise_speed	`Vector3(0, 0, 0)`
float	turbulence_noise_speed_random	`0.2`

float	turbulence_noise_strength	`1.0`
Texture2D	velocity_limit_curve	
Vector3	velocity_pivot	`Vector3(0, 0, 0)`

Methods

Vector2	get_param(param: Parameter) const
float	get_param_max(param: Parameter) const
float	get_param_min(param: Parameter) const
Texture2D	get_param_texture(param: Parameter) const
bool	get_particle_flag(particle_flag: ParticleFlags) const
void	set_param(param: Parameter, value: Vector2)
void	set_param_max(param: Parameter, value: float)
void	set_param_min(param: Parameter, value: float)

void	set_param_texture(param: Parameter, texture: Texture2D)
void	set_particle_flag(particle_flag: ParticleFlags, enable: bool)

Enumerations

enum **Parameter**: 🔗

Parameter **PARAM_INITIAL_LINEAR_VELOCITY** = `0`

Use with set_param_min, set_param_max, and set_param_texture to set initial velocity properties.

Parameter **PARAM_ANGULAR_VELOCITY** = `1`

Use with set_param_min, set_param_max, and set_param_texture to set angular velocity properties.

Parameter **PARAM_ORBIT_VELOCITY** = `2`

Use with set_param_min, set_param_max, and set_param_texture to set orbital velocity properties.

Parameter **PARAM_LINEAR_ACCEL** = `3`

Use with set_param_min, set_param_max, and set_param_texture to set linear acceleration properties.

Parameter **PARAM_RADIAL_ACCEL** = `4`

Use with set_param_min, set_param_max, and set_param_texture to set radial acceleration properties.

Parameter **PARAM_TANGENTIAL_ACCEL** = `5`

Use with set_param_min, set_param_max, and set_param_texture to set tangential acceleration properties.

Parameter **PARAM_DAMPING** = `6`

Use with set_param_min, set_param_max, and set_param_texture to set damping properties.

Parameter **PARAM_ANGLE** = `7`

Use with set_param_min, set_param_max, and set_param_texture to set angle properties.

Parameter **PARAM_SCALE** = `8`

Use with set_param_min, set_param_max, and set_param_texture to set scale properties.

Parameter **PARAM_HUE_VARIATION** = `9`

Use with set_param_min, set_param_max, and set_param_texture to set hue variation properties.

Parameter **PARAM_ANIM_SPEED** = `10`

Use with set_param_min, set_param_max, and set_param_texture to set animation speed properties.

Parameter **PARAM_ANIM_OFFSET** = `11`

Use with set_param_min, set_param_max, and set_param_texture to set animation offset properties.

Parameter **PARAM_RADIAL_VELOCITY** = `15`

Use with set_param_min, set_param_max, and set_param_texture to set radial velocity properties.

Parameter **PARAM_DIRECTIONAL_VELOCITY** = `16`

Use with set_param_min, set_param_max, and set_param_texture to set directional velocity properties.

Parameter **PARAM_SCALE_OVER_VELOCITY** = `17`

Use with set_param_min, set_param_max, and set_param_texture to set scale over velocity properties.

Parameter **PARAM_MAX** = `18`

Represents the size of the Parameter enum.

Parameter **PARAM_TURB_VEL_INFLUENCE** = `13`

Use with set_param_min and set_param_max to set the turbulence minimum und maximum influence on each particles velocity.

Parameter **PARAM_TURB_INIT_DISPLACEMENT** = `14`

Use with set_param_min and set_param_max to set the turbulence minimum and maximum displacement of the particles spawn position.

Parameter **PARAM_TURB_INFLUENCE_OVER_LIFE** = `12`

Use with set_param_texture to set the turbulence influence over the particles life time.

---

enum **ParticleFlags**: 🔗

ParticleFlags **PARTICLE_FLAG_ALIGN_Y_TO_VELOCITY** = `0`

Use with set_particle_flag to set particle_flag_align_y.

ParticleFlags **PARTICLE_FLAG_ROTATE_Y** = `1`

Use with set_particle_flag to set particle_flag_rotate_y.

ParticleFlags **PARTICLE_FLAG_DISABLE_Z** = `2`

Use with set_particle_flag to set particle_flag_disable_z.

ParticleFlags **PARTICLE_FLAG_DAMPING_AS_FRICTION** = `3`

There is currently no description for this enum. Please help us by contributing one!

ParticleFlags **PARTICLE_FLAG_MAX** = `4`

Represents the size of the ParticleFlags enum.

---

enum **EmissionShape**: 🔗

EmissionShape **EMISSION_SHAPE_POINT** = `0`

All particles will be emitted from a single point.

EmissionShape **EMISSION_SHAPE_SPHERE** = `1`

Particles will be emitted in the volume of a sphere.

EmissionShape **EMISSION_SHAPE_SPHERE_SURFACE** = `2`

Particles will be emitted on the surface of a sphere.

EmissionShape **EMISSION_SHAPE_BOX** = `3`

Particles will be emitted in the volume of a box.

EmissionShape **EMISSION_SHAPE_POINTS** = `4`

Particles will be emitted at a position determined by sampling a random point on the emission_point_texture. Particle color will be modulated by emission_color_texture.

EmissionShape **EMISSION_SHAPE_DIRECTED_POINTS** = `5`

Particles will be emitted at a position determined by sampling a random point on the emission_point_texture. Particle velocity and rotation will be set based on emission_normal_texture. Particle color will be modulated by emission_color_texture.

EmissionShape **EMISSION_SHAPE_RING** = `6`

Particles will be emitted in a ring or cylinder.

EmissionShape **EMISSION_SHAPE_MAX** = `7`

Represents the size of the EmissionShape enum.

enum **SubEmitterMode**: 🔗

SubEmitterMode **SUB_EMITTER_DISABLED** = `0`

There is currently no description for this enum. Please help us by contributing one!

SubEmitterMode **SUB_EMITTER_CONSTANT** = `1`

There is currently no description for this enum. Please help us by contributing one!

SubEmitterMode **SUB_EMITTER_AT_END** = `2`

There is currently no description for this enum. Please help us by contributing one!

SubEmitterMode **SUB_EMITTER_AT_COLLISION** = `3`

There is currently no description for this enum. Please help us by contributing one!

SubEmitterMode **SUB_EMITTER_MAX** = `4`

Represents the size of the SubEmitterMode enum.

enum **CollisionMode**: 🔗

CollisionMode **COLLISION_DISABLED** = `0`

No collision for particles. Particles will go through GPUParticlesCollision3D nodes.

CollisionMode **COLLISION_RIGID** = `1`

RigidBody3D-style collision for particles using GPUParticlesCollision3D nodes.

CollisionMode **COLLISION_HIDE_ON_CONTACT** = `2`

Hide particles instantly when colliding with a GPUParticlesCollision3D node. This can be combined with a subemitter that uses the COLLISION_RIGID collision mode to "replace" the parent particle with the subemitter on impact.

CollisionMode **COLLISION_MAX** = `3`

Represents the size of the CollisionMode enum.

Property Descriptions

Texture2D **alpha_curve** 🔗

- void **set_alpha_curve**(value: Texture2D)
- Texture2D **get_alpha_curve**()

The alpha value of each particle's color will be multiplied by this CurveTexture over its lifetime.

Texture2D **angle_curve** 🔗

- void **set_param_texture**(param: Parameter, texture: Texture2D)
- Texture2D **get_param_texture**(param: Parameter) const

Each particle's rotation will be animated along this CurveTexture.

---

float **angle_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum initial rotation applied to each particle, in degrees.

Only applied when particle_flag_disable_z or particle_flag_rotate_y are `true` or the BaseMaterial3D being used to draw the particle is using BaseMaterial3D.BILLBOARD_PARTICLES.

---

float **angle_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of angle_max.

Texture2D **angular_velocity_curve**

- void **set_param_texture**(param: Parameter, texture: Texture2D)
- Texture2D **get_param_texture**(param: Parameter) const

Each particle's angular velocity (rotation speed) will vary along this CurveTexture over its lifetime.

---

float **angular_velocity_max** = `0.0`

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum initial angular velocity (rotation speed) applied to each particle in *degrees* per second.

Only applied when particle_flag_disable_z or particle_flag_rotate_y are `true` or the BaseMaterial3D being used to draw the particle is using BaseMaterial3D.BILLBOARD_PARTICLES.

---

float **angular_velocity_min** = `0.0`

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of angular_velocity_max.

Texture2D **anim_offset_curve**

- void **set_param_texture**(param: Parameter, texture: Texture2D)
- Texture2D **get_param_texture**(param: Parameter) const

Each particle's animation offset will vary along this CurveTexture.

Float **anim_offset_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum animation offset that corresponds to frame index in the texture. `0` is the first frame, `1` is the last one. See CanvasItemMaterial.particles_animation.

---

float **anim_offset_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of anim_offset_max.

Texture2D **anim_speed_curve** 🔗

- void **set_param_texture**(param: Parameter, texture: Texture2D)
- Texture2D **get_param_texture**(param: Parameter) const

Each particle's animation speed will vary along this CurveTexture.

---

float **anim_speed_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)

- float **get_param_max**(param: Parameter) const

Maximum particle animation speed. Animation speed of `1` means that the particles will make full `0` to `1` offset cycle during lifetime, `2` means `2` cycles etc.

With animation speed greater than `1`, remember to enable CanvasItemMaterial.particles_anim_loop property if you want the animation to repeat.

float **anim_speed_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of anim_speed_max.

bool **attractor_interaction_enabled** = `true` 🔗

- void **set_attractor_interaction_enabled**(value: bool)
- bool **is_attractor_interaction_enabled**()

If `true`, interaction with particle attractors is enabled. In 3D, attraction only occurs within the area defined by the GPUParticles3D node's GPUParticles3D.visibility_aabb.

float **collision_bounce** 🔗

- void **set_collision_bounce**(value: float)
- float **get_collision_bounce**()

The particles' bounciness. Values range from `0` (no bounce) to `1` (full bounciness). Only effective if collision_mode is COLLISION_RIGID.

float **collision_friction** 🔗

- void **set_collision_friction**(value: float)
- float **get_collision_friction**()

The particles' friction. Values range from `0` (frictionless) to `1` (maximum friction). Only effective if collision_mode is COLLISION_RIGID.

CollisionMode **collision_mode** = `0` 🔗

- void **set_collision_mode**(value: CollisionMode)
- CollisionMode **get_collision_mode**()

The particles' collision mode.

Note: 3D Particles can only collide with GPUParticlesCollision3D nodes, not PhysicsBody3D nodes. To make particles collide with various objects, you can add GPUParticlesCollision3D nodes as children of PhysicsBody3D nodes. In 3D, collisions only occur within the area defined by the GPUParticles3D node's GPUParticles3D.visibility_aabb.

Note: 2D Particles can only collide with LightOccluder2D nodes, not PhysicsBody2D nodes.

---

bool **collision_use_scale** = `false` 🔗

- void **set_collision_use_scale**(value: bool)
- bool **is_collision_using_scale**()

If `true`, GPUParticles3D.collision_base_size is multiplied by the particle's effective scale (see scale_min, scale_max, scale_curve, and scale_over_velocity_curve).

Color **color** = `Color(1, 1, 1, 1)` 🔗

- void **set_color**(value: Color)
- Color **get_color**()

Each particle's initial color. If the GPUParticles2D's `texture` is defined, it will be multiplied by this color.

Note: color multiplies the particle mesh's vertex colors. To have a visible effect on a BaseMaterial3D, BaseMaterial3D.vertex_color_use_as_albedo *must* be `true`. For a ShaderMaterial, `ALBEDO *= COLOR.rgb;` must be inserted in the shader's `fragment()` function. Otherwise, color will have no visible effect.

Texture2D **color_initial_ramp** 🔗

- void **set_color_initial_ramp**(value: Texture2D)
- Texture2D **get_color_initial_ramp**()

Each particle's initial color will vary along this GradientTexture1D (multiplied with color).

Note: color_initial_ramp multiplies the particle mesh's vertex colors. To have a visible effect on a BaseMaterial3D, BaseMaterial3D.vertex_color_use_as_albedo *must* be `true`. For a ShaderMaterial, `ALBEDO *= COLOR.rgb;` must be inserted in the shader's `fragment()` function. Otherwise, color_initial_ramp will have no visible effect.

Texture2D **color_ramp** 🔗

- void **set_color_ramp**(value: Texture2D)
- Texture2D **get_color_ramp**()

Each particle's color will vary along this GradientTexture1D over its lifetime (multiplied with color).

Note: color_ramp multiplies the particle mesh's vertex colors. To have a visible effect on a BaseMaterial3D, BaseMaterial3D.vertex_color_use_as_albedo *must* be `true`. For a ShaderMaterial, `ALBEDO *= COLOR.rgb;` must be inserted in the shader's `fragment()` function. Otherwise, color_ramp will have no visible effect.

Texture2D **damping_curve** 🔗

- void **set_param_texture**(param: Parameter, texture: Texture2D)
- Texture2D **get_param_texture**(param: Parameter) const

Damping will vary along this CurveTexture.

---

float **damping_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

The maximum rate at which particles lose velocity. For example value of `100` means that the particle will go from `100` velocity to `0` in `1` second.

---

float **damping_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of damping_max.

Vector3 **direction** = `Vector3(1, 0, 0)`

- void **set_direction**(value: Vector3)
- Vector3 **get_direction**()

Unit vector specifying the particles' emission direction.

Texture2D **directional_velocity_curve**

- void **set_param_texture**(param: Parameter, texture: Texture2D)
- Texture2D **get_param_texture**(param: Parameter) const

A curve that specifies the velocity along each of the axes of the particle system along its lifetime.

Note: Animated velocities will not be affected by damping, use velocity_limit_curve instead.

---

float **directional_velocity_max**

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum directional velocity value, which is multiplied by directional_velocity_curve.

Note: Animated velocities will not be affected by damping, use velocity_limit_curve instead.

---

float **directional_velocity_min**

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum directional velocity value, which is multiplied by directional_velocity_curve.

Note: Animated velocities will not be affected by damping, use velocity_limit_curve instead.

Vector3 **emission_box_extents**

- void **set_emission_box_extents**(value: Vector3)
- Vector3 **get_emission_box_extents**()

The box's extents if emission_shape is set to EMISSION_SHAPE_BOX.

Note: emission_box_extents starts from the center point and applies the X, Y, and Z values in both directions. The size is twice the area of the extents.

Texture2D **emission_color_texture**

- void **set_emission_color_texture**(value: Texture2D)
- Texture2D **get_emission_color_texture**()

Particle color will be modulated by color determined by sampling this texture at the same point as the emission_point_texture.

Note: emission_color_texture multiplies the particle mesh's vertex colors. To have a visible effect on a BaseMaterial3D, BaseMaterial3D.vertex_color_use_as_albedo *must* be `true`. For a ShaderMaterial, `ALBEDO *= COLOR.rgb;` must be inserted in the shader's `fragment()` function. Otherwise, emission_color_texture will have no visible effect.

Texture2D **emission_curve**

- void **set_emission_curve**(value: Texture2D)
- Texture2D **get_emission_curve**()

Each particle's color will be multiplied by this CurveTexture over its lifetime.

Note: This property won't have a visible effect unless the render material is marked as unshaded.

Texture2D **emission_normal_texture**

- void **set_emission_normal_texture**(value: Texture2D)
- Texture2D **get_emission_normal_texture**()

Particle velocity and rotation will be set by sampling this texture at the same point as the emission_point_texture. Used only in EMISSION_SHAPE_DIRECTED_POINTS. Can be created automatically from mesh or node by selecting "Create Emission Points from Mesh/Node" under the "Particles" tool in the toolbar.

---

int **emission_point_count**

- void **set_emission_point_count**(value: int)
- int **get_emission_point_count**()

The number of emission points if emission_shape is set to EMISSION_SHAPE_POINTS or EMISSION_SHAPE_DIRECTED_POINTS.

Texture2D **emission_point_texture**

- void **set_emission_point_texture**(value: Texture2D)
- Texture2D **get_emission_point_texture**()

Particles will be emitted at positions determined by sampling this texture at a random position. Used with EMISSION_SHAPE_POINTS and EMISSION_SHAPE_DIRECTED_POINTS. Can be created automatically from mesh or node by selecting "Create Emission Points from Mesh/Node" under the "Particles" tool in the toolbar.

Vector3 **emission_ring_axis** 🔗

- void **set_emission_ring_axis**(value: Vector3)
- Vector3 **get_emission_ring_axis**()

The axis of the ring when using the emitter EMISSION_SHAPE_RING.

---

float **emission_ring_height** 🔗

- void **set_emission_ring_height**(value: float)
- float **get_emission_ring_height**()

The height of the ring when using the emitter EMISSION_SHAPE_RING.

---

float **emission_ring_inner_radius** 🔗

- void **set_emission_ring_inner_radius**(value: float)
- float **get_emission_ring_inner_radius**()

The inner radius of the ring when using the emitter EMISSION_SHAPE_RING.

---

float **emission_ring_radius** 🔗

- void **set_emission_ring_radius**(value: float)

- float **get_emission_ring_radius**()

The radius of the ring when using the emitter EMISSION_SHAPE_RING.

EmissionShape **emission_shape** = `0`

- void **set_emission_shape**(value: EmissionShape)
- EmissionShape **get_emission_shape**()

Particles will be emitted inside this region. Use EmissionShape constants for values.

Vector3 **emission_shape_offset** = `Vector3(0, 0, 0)`

- void **set_emission_shape_offset**(value: Vector3)
- Vector3 **get_emission_shape_offset**()

The offset for the emission_shape, in local space.

Vector3 **emission_shape_scale** = `Vector3(1, 1, 1)`

- void **set_emission_shape_scale**(value: Vector3)
- Vector3 **get_emission_shape_scale**()

The scale of the emission_shape, in local space.

---

float **emission_sphere_radius**

- void **set_emission_sphere_radius**(value: float)
- float **get_emission_sphere_radius**()

The sphere's radius if emission_shape is set to EMISSION_SHAPE_SPHERE.

---

float **flatness** = `0.0`

- void **set_flatness**(value: float)
- float **get_flatness**()

Amount of spread along the Y axis.

Vector3 **gravity** = `Vector3(0, -9.8, 0)`

- void **set_gravity**(value: Vector3)
- Vector3 **get_gravity**()

Gravity applied to every particle.

Texture2D **hue_variation_curve**

- void **set_param_texture**(param: Parameter, texture: Texture2D)
- Texture2D **get_param_texture**(param: Parameter) const

Each particle's hue will vary along this CurveTexture.

---

float **hue_variation_max** = `0.0`

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum initial hue variation applied to each particle. It will shift the particle color's hue.

---

float **hue_variation_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of hue_variation_max.

---

float **inherit_velocity_ratio** = `0.0` 🔗

- void **set_inherit_velocity_ratio**(value: float)
- float **get_inherit_velocity_ratio**()

Percentage of the velocity of the respective GPUParticles2D or GPUParticles3D inherited by each particle when spawning.

---

float **initial_velocity_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum initial velocity magnitude for each particle. Direction comes from direction and spread.

---

float **initial_velocity_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of initial_velocity_max.

---

float **lifetime_randomness** = `0.0` 🔗

- void **set_lifetime_randomness**(value: float)

- float **get_lifetime_randomness**()

Particle lifetime randomness ratio. The equation for the lifetime of a particle is `lifetime * (1.0 - randf() * lifetime_randomness)`. For example, a lifetime_randomness of `0.4` scales the lifetime between `0.6` to `1.0` of its original value.

Texture2D **linear_accel_curve** 🔗

- void **set_param_texture**(param: Parameter, texture: Texture2D)
- Texture2D **get_param_texture**(param: Parameter) const

Each particle's linear acceleration will vary along this CurveTexture.

---

float **linear_accel_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum linear acceleration applied to each particle in the direction of motion.

---

float **linear_accel_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of linear_accel_max.

Texture2D **orbit_velocity_curve** 🔗

- void **set_param_texture**(param: Parameter, texture: Texture2D)
- Texture2D **get_param_texture**(param: Parameter) const

Each particle's orbital velocity will vary along this CurveTexture.

Note: For 3D orbital velocity, use a CurveXYZTexture.

Note: Animated velocities will not be affected by damping, use velocity_limit_curve instead.

---

float **orbit_velocity_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum orbital velocity applied to each particle. Makes the particles circle around origin. Specified in number of full rotations around origin per second.

Note: Animated velocities will not be affected by damping, use velocity_limit_curve instead.

---

float **orbit_velocity_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of orbit_velocity_max.

Note: Animated velocities will not be affected by damping, use velocity_limit_curve instead.

---

bool **particle_flag_align_y** = `false` 🔗

- void **set_particle_flag**(particle_flag: ParticleFlags, enable: bool)
- bool **get_particle_flag**(particle_flag: ParticleFlags) const

Align Y axis of particle with the direction of its velocity.

---

bool **particle_flag_damping_as_friction** = `false`

- void **set_particle_flag**(particle_flag: ParticleFlags, enable: bool)
- bool **get_particle_flag**(particle_flag: ParticleFlags) const

Changes the behavior of the damping properties from a linear deceleration to a deceleration based on speed percentage.

---

bool **particle_flag_disable_z** = `false`

- void **set_particle_flag**(particle_flag: ParticleFlags, enable: bool)
- bool **get_particle_flag**(particle_flag: ParticleFlags) const

If `true`, particles will not move on the z axis.

---

bool **particle_flag_rotate_y** = `false`

- void **set_particle_flag**(particle_flag: ParticleFlags, enable: bool)
- bool **get_particle_flag**(particle_flag: ParticleFlags) const

If `true`, particles rotate around Y axis by angle_min.

Texture2D **radial_accel_curve**

- void **set_param_texture**(param: Parameter, texture: Texture2D)
- Texture2D **get_param_texture**(param: Parameter) const

Each particle's radial acceleration will vary along this CurveTexture.

float **radial_accel_max** = `0.0`

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum radial acceleration applied to each particle. Makes particle accelerate away from the origin or towards it if negative.

float **radial_accel_min** = `0.0`

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of radial_accel_max.

Texture2D **radial_velocity_curve**

- void **set_param_texture**(param: Parameter, texture: Texture2D)
- Texture2D **get_param_texture**(param: Parameter) const

A CurveTexture that defines the velocity over the particle's lifetime away (or toward) the velocity_pivot.

Note: Animated velocities will not be affected by damping, use velocity_limit_curve instead.

float **radial_velocity_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum radial velocity applied to each particle. Makes particles move away from the velocity_pivot, or toward it if negative.

Note: Animated velocities will not be affected by damping, use velocity_limit_curve instead.

---

float **radial_velocity_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum radial velocity applied to each particle. Makes particles move away from the velocity_pivot, or toward it if negative.

Note: Animated velocities will not be affected by damping, use velocity_limit_curve instead.

Texture2D **scale_curve** 🔗

- void **set_param_texture**(param: Parameter, texture: Texture2D)
- Texture2D **get_param_texture**(param: Parameter) const

Each particle's scale will vary along this CurveTexture over its lifetime. If a CurveXYZTexture is supplied instead, the scale will be separated per-axis.

---

float **scale_max** = `1.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum initial scale applied to each particle.

---

float **scale_min** = `1.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of scale_max.

Texture2D **scale_over_velocity_curve** 🔗

- void **set_param_texture**(param: Parameter, texture: Texture2D)
- Texture2D **get_param_texture**(param: Parameter) const

Either a CurveTexture or a CurveXYZTexture that scales each particle based on its velocity.

---

float **scale_over_velocity_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum velocity value reference for scale_over_velocity_curve.

scale_over_velocity_curve will be interpolated between scale_over_velocity_min and scale_over_velocity_max.

---

float **scale_over_velocity_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum velocity value reference for scale_over_velocity_curve.

scale_over_velocity_curve will be interpolated between scale_over_velocity_min and scale_over_velocity_max.

---

float **spread** = `45.0`

- void **set_spread**(value: float)
- float **get_spread**()

Each particle's initial direction range from `+spread` to `-spread` degrees.

---

int **sub_emitter_amount_at_collision**

- void **set_sub_emitter_amount_at_collision**(value: int)
- int **get_sub_emitter_amount_at_collision**()

The amount of particles to spawn from the subemitter node when a collision occurs. When combined with COLLISION_HIDE_ON_CONTACT on the main particles material, this can be used to achieve effects such as raindrops hitting the ground.

Note: This value shouldn't exceed GPUParticles2D.amount or GPUParticles3D.amount defined on the *subemitter node* (not the main node), relative to the subemitter's particle lifetime. If the number of particles is exceeded, no new particles will spawn from the subemitter until enough particles have expired.

---

int **sub_emitter_amount_at_end**

- void **set_sub_emitter_amount_at_end**(value: int)
- int **get_sub_emitter_amount_at_end**()

The amount of particles to spawn from the subemitter node when the particle expires.

Note: This value shouldn't exceed GPUParticles2D.amount or GPUParticles3D.amount defined on the *subemitter node* (not the main node), relative to the subemitter's particle lifetime. If the number of particles is exceeded, no new particles will spawn from the subemitter until enough particles have expired.

---

float **sub_emitter_frequency** 🔗

- void **set_sub_emitter_frequency**(value: float)
- float **get_sub_emitter_frequency**()

The frequency at which particles should be emitted from the subemitter node. One particle will be spawned every sub_emitter_frequency seconds.

Note: This value shouldn't exceed GPUParticles2D.amount or GPUParticles3D.amount defined on the *subemitter node* (not the main node), relative to the subemitter's particle lifetime. If the number of particles is exceeded, no new particles will spawn from the subemitter until enough particles have expired.

---

bool **sub_emitter_keep_velocity** = `false` 🔗

- void **set_sub_emitter_keep_velocity**(value: bool)
- bool **get_sub_emitter_keep_velocity**()

If `true`, the subemitter inherits the parent particle's velocity when it spawns.

SubEmitterMode **sub_emitter_mode** = `0` 🔗

- void **set_sub_emitter_mode**(value: SubEmitterMode)
- SubEmitterMode **get_sub_emitter_mode**()

The particle subemitter mode (see GPUParticles2D.sub_emitter and GPUParticles3D.sub_emitter).

Texture2D **tangential_accel_curve** 🔗

- void **set_param_texture**(param: Parameter, texture: Texture2D)
- Texture2D **get_param_texture**(param: Parameter) const

Each particle's tangential acceleration will vary along this CurveTexture.

---

float **tangential_accel_max** = `0.0` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum tangential acceleration applied to each particle. Tangential acceleration is perpendicular to the particle's velocity giving the particles a swirling motion.

---

float **tangential_accel_min** = `0.0` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum equivalent of tangential_accel_max.

---

bool **turbulence_enabled** = `false` 🔗

- void **set_turbulence_enabled**(value: bool)
- bool **get_turbulence_enabled**()

If `true`, enables turbulence for the particle system. Turbulence can be used to vary particle movement according to its position (based on a 3D noise pattern). In 3D, GPUParticlesAttractorVectorField3D with NoiseTexture3D can be used as an alternative to turbulence that works in world space and with multiple particle systems reacting in the same way.

Note: Enabling turbulence has a high performance cost on the GPU. Only enable turbulence on a few particle systems at once at most, and consider disabling it when targeting mobile/web platforms.

---

float **turbulence_influence_max** = `0.1` 🔗

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum turbulence influence on each particle.

The actual amount of turbulence influence on each particle is calculated as a random value between turbulence_influence_min and turbulence_influence_max and multiplied by the amount of turbulence influence from turbulence_influence_over_life.

---

float **turbulence_influence_min** = `0.1` 🔗

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum turbulence influence on each particle.

The actual amount of turbulence influence on each particle is calculated as a random value between turbulence_influence_min and turbulence_influence_max and multiplied by the amount of turbulence influence from turbulence_influence_over_life.

Texture2D **turbulence_influence_over_life**

- void **set_param_texture**(param: Parameter, texture: Texture2D)
- Texture2D **get_param_texture**(param: Parameter) const

Each particle's amount of turbulence will be influenced along this CurveTexture over its life time.

---

float **turbulence_initial_displacement_max** = `0.0`

- void **set_param_max**(param: Parameter, value: float)
- float **get_param_max**(param: Parameter) const

Maximum displacement of each particle's spawn position by the turbulence.

The actual amount of displacement will be a factor of the underlying turbulence multiplied by a random value between turbulence_initial_displacement_min and turbulence_initial_displacement_max.

---

float **turbulence_initial_displacement_min** = `0.0`

- void **set_param_min**(param: Parameter, value: float)
- float **get_param_min**(param: Parameter) const

Minimum displacement of each particle's spawn position by the turbulence.

The actual amount of displacement will be a factor of the underlying turbulence multiplied by a random value between turbulence_initial_displacement_min and turbulence_initial_displacement_max.

---

float **turbulence_noise_scale** = `9.0` 🔗

- void **set_turbulence_noise_scale**(value: float)
- float **get_turbulence_noise_scale**()

This value controls the overall scale/frequency of the turbulence noise pattern.

A small scale will result in smaller features with more detail while a high scale will result in smoother noise with larger features.

Vector3 **turbulence_noise_speed** = `Vector3(0, 0, 0)` 🔗

- void **set_turbulence_noise_speed**(value: Vector3)
- Vector3 **get_turbulence_noise_speed**()

A scrolling velocity for the turbulence field. This sets a directional trend for the pattern to move in over time.

The default value of `Vector3(0, 0, 0)` turns off the scrolling.

---

float **turbulence_noise_speed_random** = `0.2` 🔗

- void **set_turbulence_noise_speed_random**(value: float)
- float **get_turbulence_noise_speed_random**()

The in-place rate of change of the turbulence field. This defines how quickly the noise pattern varies over time.

A value of 0.0 will result in a fixed pattern.

float **turbulence_noise_strength** = `1.0`

- void **set_turbulence_noise_strength**(value: float)
- float **get_turbulence_noise_strength**()

The turbulence noise strength. Increasing this will result in a stronger, more contrasting, flow pattern.

Texture2D **velocity_limit_curve**

- void **set_velocity_limit_curve**(value: Texture2D)
- Texture2D **get_velocity_limit_curve**()

A CurveTexture that defines the maximum velocity of a particle during its lifetime.

Vector3 **velocity_pivot** = `Vector3(0, 0, 0)`

- void **set_velocity_pivot**(value: Vector3)
- Vector3 **get_velocity_pivot**()

A pivot point used to calculate radial and orbital velocity of particles.

Method Descriptions

Vector2 **get_param**(param: Parameter) const

Returns the minimum and maximum values of the given `param` as a vector.

The `x` component of the returned vector corresponds to minimum and the `y` component corresponds to maximum.

---

float **get_param_max**(param: Parameter) const

Returns the maximum value range for the given parameter.

---

float **get_param_min**(param: Parameter) const

Returns the minimum value range for the given parameter.

Texture2D **get_param_texture**(param: Parameter) const

Returns the Texture2D used by the specified parameter.

---

bool **get_particle_flag**(particle_flag: ParticleFlags) const

Returns `true` if the specified particle flag is enabled. See ParticleFlags for options.

---

void **set_param**(param: Parameter, value: Vector2)

Sets the minimum and maximum values of the given `param`.

The `x` component of the argument vector corresponds to minimum and the `y` component corresponds to maximum.

---

void **set_param_max**(param: Parameter, value: float)

Sets the maximum value range for the given parameter.

---

void **set_param_min**(param: Parameter, value: float) 🔗

Sets the minimum value range for the given parameter.

---

void **set_param_texture**(param: Parameter, texture: Texture2D) 🔗

Sets the Texture2D for the specified Parameter.

---

void **set_particle_flag**(particle_flag: ParticleFlags, enable: bool) 🔗

If `true`, enables the specified particle flag. See ParticleFlags for options.

PhysicalSkyMaterial

Inherits: Material **<** Resource **<** RefCounted **<** Object

A material that defines a sky for a Sky resource by a set of physical properties.

Description

The **PhysicalSkyMaterial** uses the Preetham analytic daylight model to draw a sky based on physical properties. This results in a substantially more realistic sky than the ProceduralSkyMaterial, but it is slightly slower and less flexible.

The **PhysicalSkyMaterial** only supports one sun. The color, energy, and direction of the sun are taken from the first DirectionalLight3D in the scene tree.

Properties

float	energy_multiplier	`1.0`
Color	ground_color	`Color(0.1, 0.07, 0.034, 1)`
float	mie_coefficient	`0.005`
Color	mie_color	`Color(0.69, 0.729, 0.812, 1)`

float	mie_eccentricity	`0.8`
Texture2D	night_sky	
float	rayleigh_coefficient	`2.0`
Color	rayleigh_color	`Color(0.3, 0.405, 0.6, 1)`
float	sun_disk_scale	`1.0`
float	turbidity	`10.0`
bool	use_debanding	`true`

Property Descriptions

float **energy_multiplier** = `1.0` 🔗

- void **set_energy_multiplier**(value: float)
- float **get_energy_multiplier**()

The sky's overall brightness multiplier. Higher values result in a brighter sky.

Color **ground_color** = `Color(0.1, 0.07, 0.034, 1)` 🔗

- void **set_ground_color**(value: Color)

- Color **get_ground_color**()

Modulates the Color on the bottom half of the sky to represent the ground.

float **mie_coefficient** = `0.005`

- void **set_mie_coefficient**(value: float)
- float **get_mie_coefficient**()

Controls the strength of Mie scattering [https://en.wikipedia.org/wiki/Mie_scattering] for the sky. Mie scattering results from light colliding with larger particles (like water). On earth, Mie scattering results in a whitish color around the sun and horizon.

Color **mie_color** = `Color(0.69, 0.729, 0.812, 1)`

- void **set_mie_color**(value: Color)
- Color **get_mie_color**()

Controls the Color of the Mie scattering [https://en.wikipedia.org/wiki/Mie_scattering] effect. While not physically accurate, this allows for the creation of alien-looking planets.

float **mie_eccentricity** = `0.8`

- void **set_mie_eccentricity**(value: float)
- float **get_mie_eccentricity**()

Controls the direction of the Mie scattering [https://en.wikipedia.org/wiki/Mie_scattering]. A value of `1` means that when light hits a particle it's passing through straight forward. A value of `-1` means that all light is scatter backwards.

Texture2D **night_sky** 🔗

- void **set_night_sky**(value: Texture2D)
- Texture2D **get_night_sky**()

Texture2D for the night sky. This is added to the sky, so if it is bright enough, it may be visible during the day.

---

float **rayleigh_coefficient** = `2.0` 🔗

- void **set_rayleigh_coefficient**(value: float)
- float **get_rayleigh_coefficient**()

Controls the strength of the Rayleigh scattering [https://en.wikipedia.org/wiki/Rayleigh_scattering]. Rayleigh scattering results from light colliding with small particles. It is responsible for the blue color of the sky.

Color **rayleigh_color** = `Color(0.3, 0.405, 0.6, 1)` 🔗

- void **set_rayleigh_color**(value: Color)
- Color **get_rayleigh_color**()

Controls the Color of the Rayleigh scattering [https://en.wikipedia.org/wiki/Rayleigh_scattering]. While not physically accurate, this allows for the creation of alien-looking planets. For example, setting this to a red Color results in a Mars-looking atmosphere with a corresponding blue sunset.

---

float **sun_disk_scale** = `1.0` 🔗

- void **set_sun_disk_scale**(value: float)
- float **get_sun_disk_scale**()

Sets the size of the sun disk. Default value is based on Sol's perceived size from Earth.

---

float **turbidity** = `10.0`

- void **set_turbidity**(value: float)
- float **get_turbidity**()

Sets the thickness of the atmosphere. High turbidity creates a foggy-looking atmosphere, while a low turbidity results in a clearer atmosphere.

---

bool **use_debanding** = `true`

- void **set_use_debanding**(value: bool)
- bool **get_use_debanding**()

If `true`, enables debanding. Debanding adds a small amount of noise which helps reduce banding that appears from the smooth changes in color in the sky.

PhysicsMaterial

Inherits: Resource **<** RefCounted **<** Object

Holds physics-related properties of a surface, namely its roughness and bounciness.

Description

Holds physics-related properties of a surface, namely its roughness and bounciness. This class is used to apply these properties to a physics body.

Properties

bool	absorbent	`false`
float	bounce	`0.0`
float	friction	`1.0`
bool	rough	`false`

Property Descriptions

bool **absorbent** = `false` 🔗

- void **set_absorbent**(value: bool)
- bool **is_absorbent**()

If `true`, subtracts the bounciness from the colliding object's bounciness instead of adding it.

---

float **bounce** = `0.0` 🔗

- void **set_bounce**(value: float)
- float **get_bounce**()

The body's bounciness. Values range from `0` (no bounce) to `1` (full bounciness).

Note: Even with bounce set to `1.0`, some energy will be lost over time due to linear and angular damping. To have a physics body that preserves all its energy over time, set bounce to `1.0`, the body's linear damp mode to **Replace** (if applicable), its linear damp to `0.0`, its angular damp mode to **Replace** (if applicable), and its angular damp to `0.0`.

---

float **friction** = `1.0` 🔗

- void **set_friction**(value: float)
- float **get_friction**()

The body's friction. Values range from `0` (frictionless) to `1` (maximum friction).

---

bool **rough** = `false` 🔗

- void **set_rough**(value: bool)

- bool **is_rough**()

If `true`, the physics engine will use the friction of the object marked as "rough" when two objects collide. If `false`, the physics engine will use the lowest friction of all colliding objects instead. If `true` for both colliding objects, the physics engine will use the highest friction.

PlaceholderCubemap

Inherits: PlaceholderTextureLayered **<** TextureLayered **<** Texture **<** Resource **<** RefCounted **<** Object

A Cubemap without image data.

Description

This class replaces a Cubemap or a Cubemap-derived class in 2 conditions:

- In dedicated server mode, where the image data shouldn't affect game logic. This allows reducing the exported PCK's size significantly.
- When the Cubemap-derived class is missing, for example when using a different engine version.

Note: This class is not intended for rendering or for use in shaders. Operations like calculating UV are not guaranteed to work.

PlaceholderCubemapArray

Inherits: PlaceholderTextureLayered **<** TextureLayered **<** Texture **<** Resource **<** RefCounted **<** Object

A CubemapArray without image data.

Description

This class replaces a CubemapArray or a CubemapArray-derived class in 2 conditions:

- In dedicated server mode, where the image data shouldn't affect game logic. This allows reducing the exported PCK's size significantly.
- When the CubemapArray-derived class is missing, for example when using a different engine version.

Note: This class is not intended for rendering or for use in shaders. Operations like calculating UV are not guaranteed to work.

PlaceholderMaterial

Inherits: Material **<** Resource **<** RefCounted **<** Object

Placeholder class for a material.

Description

This class is used when loading a project that uses a Material subclass in 2 conditions:

- When running the project exported in dedicated server mode, only the texture's dimensions are kept (as they may be relied upon for gameplay purposes or positioning of other elements). This allows reducing the exported PCK's size significantly.
- When this subclass is missing due to using a different engine version or build (e.g. modules disabled).

PlaceholderMesh

Inherits: Mesh **<** Resource **<** RefCounted **<** Object

Placeholder class for a mesh.

Description

This class is used when loading a project that uses a Mesh subclass in 2 conditions:

- When running the project exported in dedicated server mode, only the texture's dimensions are kept (as they may be relied upon for gameplay purposes or positioning of other elements). This allows reducing the exported PCK's size significantly.
- When this subclass is missing due to using a different engine version or build (e.g. modules disabled).

Properties

AABB	aabb	`AABB(0, 0, 0, 0, 0, 0)`

Property Descriptions

AABB **aabb** = `AABB(0, 0, 0, 0, 0, 0)`

- void **set_aabb**(value: AABB)
- AABB **get_aabb**()

The smallest AABB enclosing this mesh in local space.

PlaceholderTexture2D

Inherits: Texture2D **<** Texture **<** Resource **<** RefCounted **<** Object

Placeholder class for a 2-dimensional texture.

Description

This class is used when loading a project that uses a Texture2D subclass in 2 conditions:

- When running the project exported in dedicated server mode, only the texture's dimensions are kept (as they may be relied upon for gameplay purposes or positioning of other elements). This allows reducing the exported PCK's size significantly.
- When this subclass is missing due to using a different engine version or build (e.g. modules disabled).

Note: This is not intended to be used as an actual texture for rendering. It is not guaranteed to work like one in shaders or materials (for example when calculating UV).

Properties

bool	resource_local_to_scene	`false` (overrides Resource)
Vector2	size	`Vector2(1, 1)`

Property Descriptions

Vector2 **size** = `Vector2(1, 1)` 🔗

- void **set_size**(value: Vector2)
- Vector2 **get_size**()

The texture's size (in pixels).

PlaceholderTexture2DArray

Inherits: PlaceholderTextureLayered **<** TextureLayered **<** Texture **<** Resource **<** RefCounted **<** Object

Placeholder class for a 2-dimensional texture array.

Description

This class is used when loading a project that uses a Texture2D subclass in 2 conditions:

- When running the project exported in dedicated server mode, only the texture's dimensions are kept (as they may be relied upon for gameplay purposes or positioning of other elements). This allows reducing the exported PCK's size significantly.
- When this subclass is missing due to using a different engine version or build (e.g. modules disabled).

Note: This is not intended to be used as an actual texture for rendering. It is not guaranteed to work like one in shaders or materials (for example when calculating UV).

PlaceholderTexture3D

Inherits: Texture3D **<** Texture **<** Resource **<** RefCounted **<** Object

Placeholder class for a 3-dimensional texture.

Description

This class is used when loading a project that uses a Texture3D subclass in 2 conditions:

- When running the project exported in dedicated server mode, only the texture's dimensions are kept (as they may be relied upon for gameplay purposes or positioning of other elements). This allows reducing the exported PCK's size significantly.
- When this subclass is missing due to using a different engine version or build (e.g. modules disabled).

Note: This is not intended to be used as an actual texture for rendering. It is not guaranteed to work like one in shaders or materials (for example when calculating UV).

Properties

Vector3i	size	`Vector3i(1, 1, 1)`

Property Descriptions

Vector3i **size** = `Vector3i(1, 1, 1)`

- void **set_size**(value: Vector3i)
- Vector3i **get_size**()

The texture's size (in pixels).

PlaceholderTextureLayered

Inherits: TextureLayered **<** Texture **<** Resource **<** RefCounted **<** Object

Inherited By: PlaceholderCubemap, PlaceholderCubemapArray, PlaceholderTexture2DArray

Placeholder class for a 2-dimensional texture array.

Description

This class is used when loading a project that uses a TextureLayered subclass in 2 conditions:

- When running the project exported in dedicated server mode, only the texture's dimensions are kept (as they may be relied upon for gameplay purposes or positioning of other elements). This allows reducing the exported PCK's size significantly.
- When this subclass is missing due to using a different engine version or build (e.g. modules disabled).

Note: This is not intended to be used as an actual texture for rendering. It is not guaranteed to work like one in shaders or materials (for example when calculating UV).

Properties

int	layers	`1`

Vector2i	size	`Vector2i(1, 1)`

Property Descriptions

int **layers** = `1`

- void **set_layers**(value: int)
- int **get_layers**()

The number of layers in the texture array.

Vector2i **size** = `Vector2i(1, 1)`

- void **set_size**(value: Vector2i)
- Vector2i **get_size**()

The size of each texture layer (in pixels).

PlaneMesh

Inherits: PrimitiveMesh **<** Mesh **<** Resource **<** RefCounted **<** Object

Inherited By: QuadMesh

Class representing a planar PrimitiveMesh.

Description

Class representing a planar PrimitiveMesh. This flat mesh does not have a thickness. By default, this mesh is aligned on the X and Z axes; this default rotation isn't suited for use with billboarded materials. For billboarded materials, change orientation to FACE_Z.

Note: When using a large textured **PlaneMesh** (e.g. as a floor), you may stumble upon UV jittering issues depending on the camera angle. To solve this, increase subdivide_depth and subdivide_width until you no longer notice UV jittering.

Properties

Vector3	center_offset	`Vector3(0, 0, 0)`
Orientation	orientation	`1`
Vector2	size	`Vector2(2, 2)`

int	subdivide_depth	`0`
int	subdivide_width	`0`

Enumerations

enum **Orientation**: 🔗

Orientation **FACE_X** = `0`

PlaneMesh will face the positive X-axis.

Orientation **FACE_Y** = `1`

PlaneMesh will face the positive Y-axis. This matches the behavior of the **PlaneMesh** in Godot 3.x.

Orientation **FACE_Z** = `2`

PlaneMesh will face the positive Z-axis. This matches the behavior of the QuadMesh in Godot 3.x.

Property Descriptions

Vector3 **center_offset** = `Vector3(0, 0, 0)` 🔗

- void **set_center_offset**(value: Vector3)
- Vector3 **get_center_offset**()

Offset of the generated plane. Useful for particles.

Orientation **orientation** = `1` 🔗

- void **set_orientation**(value: Orientation)
- Orientation **get_orientation**()

Direction that the **PlaneMesh** is facing. See Orientation for options.

Vector2 **size** = `Vector2(2, 2)` 🔗

- void **set_size**(value: Vector2)
- Vector2 **get_size**()

Size of the generated plane.

---

int **subdivide_depth** = `0` 🔗

- void **set_subdivide_depth**(value: int)
- int **get_subdivide_depth**()

Number of subdivision along the Z axis.

---

int **subdivide_width** = `0` 🔗

- void **set_subdivide_width**(value: int)
- int **get_subdivide_width**()

Number of subdivision along the X axis.

PointMesh

Inherits: PrimitiveMesh **<** Mesh **<** Resource **<** RefCounted **<** Object

Mesh with a single Point primitive.

Description

The PointMesh is made from a single point. Instead of relying on triangles, points are rendered as a single rectangle on the screen with a constant size. They are intended to be used with Particle systems, but can be used as a cheap way to render constant size billboarded sprites (for example in a point cloud).

PointMeshes, must be used with a material that has a point size. Point size can be accessed in a shader with `POINT_SIZE`, or in a BaseMaterial3D by setting BaseMaterial3D.use_point_size and the variable BaseMaterial3D.point_size.

When using PointMeshes, properties that normally alter vertices will be ignored, including billboard mode, grow, and cull face.

PolygonOccluder3D

Inherits: Occluder3D **<** Resource **<** RefCounted **<** Object

Flat 2D polygon shape for use with occlusion culling in OccluderInstance3D.

Description

PolygonOccluder3D stores a polygon shape that can be used by the engine's occlusion culling system. When an OccluderInstance3D with a **PolygonOccluder3D** is selected in the editor, an editor will appear at the top of the 3D viewport so you can add/remove points. All points must be placed on the same 2D plane, which means it is not possible to create arbitrary 3D shapes with a single **PolygonOccluder3D**. To use arbitrary 3D shapes as occluders, use ArrayOccluder3D or OccluderInstance3D's baking feature instead.

See OccluderInstance3D's documentation for instructions on setting up occlusion culling.

Tutorials

- Occlusion culling

Properties

PackedVector2Array	polygon	`PackedVector2Array()`

Property Descriptions

PackedVector2Array **polygon** = `PackedVector2Array()`

- void **set_polygon**(value: PackedVector2Array)
- PackedVector2Array **get_polygon**()

The polygon to use for occlusion culling. The polygon can be convex or concave, but it should have as few points as possible to maximize performance.

The polygon must *not* have intersecting lines. Otherwise, triangulation will fail (with an error message printed).

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedVector2Array for more details.

PolygonPathFinder

Inherits: Resource **<** RefCounted **<** Object

There is currently no description for this class. Please help us by contributing one!

Methods

PackedVector2Array	find_path(from: Vector2, to: Vector2)
Rect2	get_bounds() const
Vector2	get_closest_point(point: Vector2) const
PackedVector2Array	get_intersections(from: Vector2, to: Vector2) const
float	get_point_penalty(idx: int) const
bool	is_point_inside(point: Vector2) const
void	set_point_penalty(idx: int, penalty: float)
void	setup(points: PackedVector2Array, connections: PackedInt32Array)

Method Descriptions

PackedVector2Array **find_path**(from: Vector2, to: Vector2) 🔗

There is currently no description for this method. Please help us by contributing one!

Rect2 **get_bounds**() const 🔗

There is currently no description for this method. Please help us by contributing one!

Vector2 **get_closest_point**(point: Vector2) const 🔗

There is currently no description for this method. Please help us by contributing one!

PackedVector2Array **get_intersections**(from: Vector2, to: Vector2) const 🔗

There is currently no description for this method. Please help us by contributing one!

---

float **get_point_penalty**(idx: int) const 🔗

There is currently no description for this method. Please help us by contributing one!

bool **is_point_inside**(point: Vector2) const 🔗

There is currently no description for this method. Please help us by contributing one!

---

void **set_point_penalty**(idx: int, penalty: float) 🔗

There is currently no description for this method. Please help us by contributing one!

---

void **setup**(points: PackedVector2Array, connections: PackedInt32Array) 🔗

There is currently no description for this method. Please help us by contributing one!

PortableCompressedTexture2D

Inherits: Texture2D **<** Texture **<** Resource **<** RefCounted **<** Object

Provides a compressed texture for disk and/or VRAM in a way that is portable.

Description

This class allows storing compressed textures as self contained (not imported) resources.

For 2D usage (compressed on disk, uncompressed on VRAM), the lossy and lossless modes are recommended. For 3D usage (compressed on VRAM) it depends on the target platform.

If you intend to only use desktop, S3TC or BPTC are recommended. For only mobile, ETC2 is recommended.

For portable, self contained 3D textures that work on both desktop and mobile, Basis Universal is recommended (although it has a small quality cost and longer compression time as a tradeoff).

This resource is intended to be created from code.

Properties

PackedByteArray	_data	`PackedByteArray()`

bool	keep_compressed_buffer	`false`
bool	resource_local_to_scene	`false` (overrides Resource)
Vector2	size_override	`Vector2(0, 0)`

Methods

void	create_from_image(image: Image, compression_mode: CompressionMode, normal_map: bool = false, lossy_quality: float = 0.8)
CompressionMode	get_compression_mode() const
Format	get_format() const
bool	is_keeping_all_compressed_buffers() static
void	set_keep_all_compressed_buffers(keep: bool) static

Enumerations

enum **CompressionMode**: 🔗

CompressionMode **COMPRESSION_MODE_LOSSLESS** = `0`

There is currently no description for this enum. Please help us by contributing one!

CompressionMode **COMPRESSION_MODE_LOSSY** = `1`

There is currently no description for this enum. Please help us by contributing one!

CompressionMode **COMPRESSION_MODE_BASIS_UNIVERSAL** = `2`

There is currently no description for this enum. Please help us by contributing one!

CompressionMode **COMPRESSION_MODE_S3TC** = `3`

There is currently no description for this enum. Please help us by contributing one!

CompressionMode **COMPRESSION_MODE_ETC2** = `4`

There is currently no description for this enum. Please help us by contributing one!

CompressionMode **COMPRESSION_MODE_BPTC** = `5`

There is currently no description for this enum. Please help us by contributing one!

Property Descriptions

PackedByteArray **_data** = `PackedByteArray()` 🔗

There is currently no description for this property. Please help us by contributing one!

---

bool **keep_compressed_buffer** = `false` 🔗

- void **set_keep_compressed_buffer**(value: bool)
- bool **is_keeping_compressed_buffer**()

When running on the editor, this class will keep the source compressed data in memory. Otherwise, the source compressed data is lost after loading and the resource can't be re saved.

This flag allows to keep the compressed data in memory if you intend it to persist after loading.

Vector2 **size_override** = `Vector2(0, 0)` 🔗

- void **set_size_override**(value: Vector2)
- Vector2 **get_size_override**()

Allow overriding the texture size (for 2D only).

Method Descriptions

void **create_from_image**(image: Image, compression_mode: CompressionMode, normal_map: bool = false, lossy_quality: float = 0.8) 🔗

Initializes the compressed texture from a base image. The compression mode must be provided.

`normal_map` is recommended to ensure optimum quality if this image will be used as a normal map.

If lossy compression is requested, the quality setting can optionally be provided. This maps to Lossy WebP compression quality.

CompressionMode **get_compression_mode**() const

Return the compression mode used (valid after initialized).

Format **get_format**() const

Return the image format used (valid after initialized).

---

bool **is_keeping_all_compressed_buffers**() static

Return whether the flag is overridden for all textures of this type.

---

void **set_keep_all_compressed_buffers**(keep: bool) static

Overrides the flag globally for all textures of this type. This is used primarily by the editor.

PrimitiveMesh

Inherits: Mesh < Resource < RefCounted < Object

Inherited By: BoxMesh, CapsuleMesh, CylinderMesh, PlaneMesh, PointMesh, PrismMesh, RibbonTrailMesh, SphereMesh, TextMesh, TorusMesh, TubeTrailMesh

Base class for all primitive meshes. Handles applying a Material to a primitive mesh.

Description

Base class for all primitive meshes. Handles applying a Material to a primitive mesh. Examples include BoxMesh, CapsuleMesh, CylinderMesh, PlaneMesh, PrismMesh, and SphereMesh.

Properties

bool	add_uv2	`false`
AABB	custom_aabb	`AABB(0, 0, 0, 0, 0, 0)`
bool	flip_faces	`false`
Material	material	

float	uv2_padding	`2.0`

Methods

Array	_create_mesh_array() virtual const
Array	get_mesh_arrays() const
void	request_update()

Property Descriptions

bool **add_uv2** = `false` 🔗

- void **set_add_uv2**(value: bool)
- bool **get_add_uv2**()

If set, generates UV2 UV coordinates applying a padding using the uv2_padding setting. UV2 is needed for lightmapping.

AABB **custom_aabb** = `AABB(0, 0, 0, 0, 0, 0)` 🔗

- void **set_custom_aabb**(value: AABB)
- AABB **get_custom_aabb**()

Overrides the AABB with one defined by user for use with frustum culling. Especially useful to avoid unexpected culling when using a shader to offset vertices.

---

bool **flip_faces** = `false` 🔗

- void **set_flip_faces**(value: bool)
- bool **get_flip_faces**()

If set, the order of the vertices in each triangle are reversed resulting in the backside of the mesh being drawn.

This gives the same result as using BaseMaterial3D.CULL_FRONT in BaseMaterial3D.cull_mode.

Material **material** 🔗

- void **set_material**(value: Material)
- Material **get_material**()

The current Material of the primitive mesh.

---

float **uv2_padding** = `2.0` 🔗

- void **set_uv2_padding**(value: float)
- float **get_uv2_padding**()

If add_uv2 is set, specifies the padding in pixels applied along seams of the mesh. Lower padding values allow making better use of the lightmap texture (resulting in higher texel density), but may introduce visible lightmap bleeding along edges.

If the size of the lightmap texture can't be determined when generating the mesh, UV2 is calculated assuming a texture size of 1024x1024.

Method Descriptions

Array **_create_mesh_array**() virtual const

Override this method to customize how this primitive mesh should be generated. Should return an Array where each element is another Array of values required for the mesh (see the ArrayType constants).

Array **get_mesh_arrays**() const

Returns mesh arrays used to constitute surface of Mesh. The result can be passed to ArrayMesh.add_surface_from_arrays to create a new surface. For example:

GDScript

```
var c = CylinderMesh.new()
var arr_mesh = ArrayMesh.new()
arr_mesh.add_surface_from_arrays(Mesh.PRIMITIVE_TRIANGLES,
c.get_mesh_arrays())
```

C#

```
var c = new CylinderMesh();
var arrMesh = new ArrayMesh();
arrMesh.AddSurfaceFromArrays(Mesh.PrimitiveType.Triangles,
c.GetMeshArrays());
```

---

void **request_update**()

Request an update of this primitive mesh based on its properties.

PrismMesh

Inherits: PrimitiveMesh **<** Mesh **<** Resource **<** RefCounted **<** Object

Class representing a prism-shaped PrimitiveMesh.

Description

Class representing a prism-shaped PrimitiveMesh.

Properties

float	left_to_right	`0.5`
Vector3	size	`Vector3(1, 1, 1)`
int	subdivide_depth	`0`
int	subdivide_height	`0`
int	subdivide_width	`0`

Property Descriptions

float **left_to_right** = `0.5`

- void **set_left_to_right**(value: float)
- float **get_left_to_right**()

Displacement of the upper edge along the X axis. 0.0 positions edge straight above the bottom-left edge.

Vector3 **size** = `Vector3(1, 1, 1)`

- void **set_size**(value: Vector3)
- Vector3 **get_size**()

Size of the prism.

---

int **subdivide_depth** = `0`

- void **set_subdivide_depth**(value: int)
- int **get_subdivide_depth**()

Number of added edge loops along the Z axis.

---

int **subdivide_height** = `0`

- void **set_subdivide_height**(value: int)
- int **get_subdivide_height**()

Number of added edge loops along the Y axis.

---

int **subdivide_width** = `0`

- void **set_subdivide_width**(value: int)
- int **get_subdivide_width**()

Number of added edge loops along the X axis.

ProceduralSkyMaterial

Inherits: Material **<** Resource **<** RefCounted **<** Object

A material that defines a simple sky for a Sky resource.

Description

ProceduralSkyMaterial provides a way to create an effective background quickly by defining procedural parameters for the sun, the sky and the ground. The sky and ground are defined by a main color, a color at the horizon, and an easing curve to interpolate between them. Suns are described by a position in the sky, a color, and a max angle from the sun at which the easing curve ends. The max angle therefore defines the size of the sun in the sky.

ProceduralSkyMaterial supports up to 4 suns, using the color, and energy, direction, and angular distance of the first four DirectionalLight3D nodes in the scene. This means that the suns are defined individually by the properties of their corresponding DirectionalLight3Ds and globally by sun_angle_max and sun_curve.

ProceduralSkyMaterial uses a lightweight shader to draw the sky and is therefore suited for real-time updates. This makes it a great option for a sky that is simple and computationally cheap, but unrealistic. If you need a more realistic procedural option, use PhysicalSkyMaterial.

Properties

float	energy_multiplier	`1.0`

Color	ground_bottom_color	`Color(0.2, 0.169, 0.133, 1)`
float	ground_curve	`0.02`
float	ground_energy_multiplier	`1.0`
Color	ground_horizon_color	`Color(0.6463, 0.6558, 0.6708, 1)`
Texture2D	sky_cover	
Color	sky_cover_modulate	`Color(1, 1, 1, 1)`
float	sky_curve	`0.15`
float	sky_energy_multiplier	`1.0`
Color	sky_horizon_color	`Color(0.6463, 0.6558, 0.6708, 1)`
Color	sky_top_color	`Color(0.385, 0.454, 0.55, 1)`
float	sun_angle_max	`30.0`

float	sun_curve	`0.15`
bool	use_debanding	`true`

Property Descriptions

float **energy_multiplier** = `1.0` 🔗

- void **set_energy_multiplier**(value: float)
- float **get_energy_multiplier**()

The sky's overall brightness multiplier. Higher values result in a brighter sky.

Color **ground_bottom_color** = `Color(0.2, 0.169, 0.133, 1)` 🔗

- void **set_ground_bottom_color**(value: Color)
- Color **get_ground_bottom_color**()

Color of the ground at the bottom. Blends with ground_horizon_color.

float **ground_curve** = `0.02` 🔗

- void **set_ground_curve**(value: float)
- float **get_ground_curve**()

How quickly the ground_horizon_color fades into the ground_bottom_color.

float **ground_energy_multiplier** = `1.0` 🔗

- void **set_ground_energy_multiplier**(value: float)
- float **get_ground_energy_multiplier**()

Multiplier for ground color. A higher value will make the ground brighter.

Color **ground_horizon_color** = `Color(0.6463, 0.6558, 0.6708, 1)` 🔗

- void **set_ground_horizon_color**(value: Color)
- Color **get_ground_horizon_color**()

Color of the ground at the horizon. Blends with ground_bottom_color.

Texture2D **sky_cover** 🔗

- void **set_sky_cover**(value: Texture2D)
- Texture2D **get_sky_cover**()

The sky cover texture to use. This texture must use an equirectangular projection (similar to PanoramaSkyMaterial). The texture's colors will be *added* to the existing sky color, and will be multiplied by sky_energy_multiplier and sky_cover_modulate. This is mainly suited to displaying stars at night, but it can also be used to display clouds at day or night (with a non-physically-accurate look).

Color **sky_cover_modulate** = `Color(1, 1, 1, 1)` 🔗

- void **set_sky_cover_modulate**(value: Color)
- Color **get_sky_cover_modulate**()

The tint to apply to the sky_cover texture. This can be used to change the sky cover's colors or opacity independently of the sky energy, which is useful for day/night or weather transitions. Only effective if a texture is defined in sky_cover.

---

float **sky_curve** = `0.15` 🔗

- void **set_sky_curve**(value: float)
- float **get_sky_curve**()

How quickly the sky_horizon_color fades into the sky_top_color.

---

float **sky_energy_multiplier** = `1.0` 🔗

- void **set_sky_energy_multiplier**(value: float)
- float **get_sky_energy_multiplier**()

Multiplier for sky color. A higher value will make the sky brighter.

Color **sky_horizon_color** = `Color(0.6463, 0.6558, 0.6708, 1)` 🔗

- void **set_sky_horizon_color**(value: Color)
- Color **get_sky_horizon_color**()

Color of the sky at the horizon. Blends with sky_top_color.

Color **sky_top_color** = `Color(0.385, 0.454, 0.55, 1)` 🔗

- void **set_sky_top_color**(value: Color)
- Color **get_sky_top_color**()

Color of the sky at the top. Blends with sky_horizon_color.

---

float **sun_angle_max** = `30.0`

- void **set_sun_angle_max**(value: float)
- float **get_sun_angle_max**()

Distance from center of sun where it fades out completely.

---

float **sun_curve** = `0.15`

- void **set_sun_curve**(value: float)
- float **get_sun_curve**()

How quickly the sun fades away between the edge of the sun disk and sun_angle_max.

---

bool **use_debanding** = `true`

- void **set_use_debanding**(value: bool)
- bool **get_use_debanding**()

If `true`, enables debanding. Debanding adds a small amount of noise which helps reduce banding that appears from the smooth changes in color in the sky.

QuadMesh

Inherits: PlaneMesh **<** PrimitiveMesh **<** Mesh **<** Resource **<** RefCounted **<** Object

Class representing a square mesh facing the camera.

Description

Class representing a square PrimitiveMesh. This flat mesh does not have a thickness. By default, this mesh is aligned on the X and Y axes; this rotation is more suited for use with billboarded materials. A **QuadMesh** is equivalent to a PlaneMesh except its default PlaneMesh.orientation is PlaneMesh.FACE_Z.

Tutorials

- GUI in 3D Viewport Demo [https://godotengine.org/asset-library/asset/2807]
- 2D in 3D Viewport Demo [https://godotengine.org/asset-library/asset/2803]

Properties

Orientation	orientation	`2` (overrides PlaneMesh)
Vector2	size	`Vector2(1, 1)` (overrides PlaneMesh)

QuadOccluder3D

Inherits: Occluder3D **<** Resource **<** RefCounted **<** Object

Flat plane shape for use with occlusion culling in OccluderInstance3D.

Description

QuadOccluder3D stores a flat plane shape that can be used by the engine's occlusion culling system. See also PolygonOccluder3D if you need to customize the quad's shape.

See OccluderInstance3D's documentation for instructions on setting up occlusion culling.

Tutorials

- Occlusion culling

Properties

Vector2	size	`Vector2(1, 1)`

Property Descriptions

Vector2 **size** = `Vector2(1, 1)` 🔗

- void **set_size**(value: Vector2)
- Vector2 **get_size**()

The quad's size in 3D units.

RDShaderFile

Inherits: Resource **<** RefCounted **<** Object

Compiled shader file in SPIR-V form (used by RenderingDevice). Not to be confused with Godot's own Shader.

Description

Compiled shader file in SPIR-V form.

See also RDShaderSource. **RDShaderFile** is only meant to be used with the RenderingDevice API. It should not be confused with Godot's own Shader resource, which is what Godot's various nodes use for high-level shader programming.

Properties

String	base_error	""

Methods

RDShaderSPIRV	get_spirv(version: StringName = &"") const
Array[StringName]	get_version_list() const

void	set_bytecode(bytecode: RDShaderSPIRV, version: StringName = &"")

Property Descriptions

String **base_error** = `""` 🔗

- void **set_base_error**(value: String)
- String **get_base_error**()

The base compilation error message, which indicates errors not related to a specific shader stage if non-empty. If empty, shader compilation is not necessarily successful (check RDShaderSPIRV's error message members).

Method Descriptions

RDShaderSPIRV **get_spirv**(version: StringName = &"") const 🔗

Returns the SPIR-V intermediate representation for the specified shader `version`.

Array[StringName] **get_version_list**() const 🔗

Returns the list of compiled versions for this shader.

---

void **set_bytecode**(bytecode: RDShaderSPIRV, version: StringName = &"") 🔗

Sets the SPIR-V `bytecode` that will be compiled for the specified `version`.

RDShaderSPIRV

Inherits: Resource **<** RefCounted **<** Object

SPIR-V intermediate representation as part of a RDShaderFile (used by RenderingDevice).

Description

RDShaderSPIRV represents a RDShaderFile's SPIR-V [https://www.khronos.org/spir/] code for various shader stages, as well as possible compilation error messages. SPIR-V is a low-level intermediate shader representation. This intermediate representation is not used directly by GPUs for rendering, but it can be compiled into binary shaders that GPUs can understand. Unlike compiled shaders, SPIR-V is portable across GPU models and driver versions.

This object is used by RenderingDevice.

Properties

PackedByteArray	bytecode_compute	`PackedByteArray()`
PackedByteArray	bytecode_fragment	`PackedByteArray()`
PackedByteArray	bytecode_tesselation_control	`PackedByteArray()`
PackedByteArray	bytecode_tesselation_evaluation	`PackedByteArray()`
PackedByteArray	bytecode_vertex	`PackedByteArray()`

String	compile_error_compute	""
String	compile_error_fragment	""
String	compile_error_tesselation_control	""
String	compile_error_tesselation_evaluation	""
String	compile_error_vertex	""

Methods

PackedByteArray	get_stage_bytecode(stage: ShaderStage) const
String	get_stage_compile_error(stage: ShaderStage) const
void	set_stage_bytecode(stage: ShaderStage, bytecode: PackedByteArray)
void	set_stage_compile_error(stage: ShaderStage, compile_error: String)

Property Descriptions

PackedByteArray **bytecode_compute** = `PackedByteArray()` 🔗

- void **set_stage_bytecode**(stage: ShaderStage, bytecode: PackedByteArray)
- PackedByteArray **get_stage_bytecode**(stage: ShaderStage) const

The SPIR-V bytecode for the compute shader stage.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedByteArray for more details.

PackedByteArray **bytecode_fragment** = `PackedByteArray()` 🔗

- void **set_stage_bytecode**(stage: ShaderStage, bytecode: PackedByteArray)
- PackedByteArray **get_stage_bytecode**(stage: ShaderStage) const

The SPIR-V bytecode for the fragment shader stage.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedByteArray for more details.

PackedByteArray **bytecode_tesselation_control** = `PackedByteArray()` 🔗

- void **set_stage_bytecode**(stage: ShaderStage, bytecode: PackedByteArray)
- PackedByteArray **get_stage_bytecode**(stage: ShaderStage) const

The SPIR-V bytecode for the tessellation control shader stage.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedByteArray for more details.

PackedByteArray **bytecode_tesselation_evaluation** = `PackedByteArray()` 🔗

- void **set_stage_bytecode**(stage: ShaderStage, bytecode: PackedByteArray)
- PackedByteArray **get_stage_bytecode**(stage: ShaderStage) const

The SPIR-V bytecode for the tessellation evaluation shader stage.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedByteArray for more details.

PackedByteArray **bytecode_vertex** = `PackedByteArray()` 🔗

- void **set_stage_bytecode**(stage: ShaderStage, bytecode: PackedByteArray)
- PackedByteArray **get_stage_bytecode**(stage: ShaderStage) const

The SPIR-V bytecode for the vertex shader stage.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedByteArray for more details.

String **compile_error_compute** = `""` 🔗

- void **set_stage_compile_error**(stage: ShaderStage, compile_error: String)
- String **get_stage_compile_error**(stage: ShaderStage) const

The compilation error message for the compute shader stage (set by the SPIR-V compiler and Godot). If empty, shader compilation was successful.

String **compile_error_fragment** = `""` 🔗

- void **set_stage_compile_error**(stage: ShaderStage, compile_error: String)
- String **get_stage_compile_error**(stage: ShaderStage) const

The compilation error message for the fragment shader stage (set by the SPIR-V compiler and Godot). If empty, shader compilation was successful.

String **compile_error_tesselation_control** = `""` 🔗

- void **set_stage_compile_error**(stage: ShaderStage, compile_error: String)
- String **get_stage_compile_error**(stage: ShaderStage) const

The compilation error message for the tessellation control shader stage (set by the SPIR-V compiler and Godot). If empty, shader compilation was successful.

String **compile_error_tesselation_evaluation** = ""

- void **set_stage_compile_error**(stage: ShaderStage, compile_error: String)
- String **get_stage_compile_error**(stage: ShaderStage) const

The compilation error message for the tessellation evaluation shader stage (set by the SPIR-V compiler and Godot). If empty, shader compilation was successful.

String **compile_error_vertex** = ""

- void **set_stage_compile_error**(stage: ShaderStage, compile_error: String)
- String **get_stage_compile_error**(stage: ShaderStage) const

The compilation error message for the vertex shader stage (set by the SPIR-V compiler and Godot). If empty, shader compilation was successful.

Method Descriptions

PackedByteArray **get_stage_bytecode**(stage: ShaderStage) const

Equivalent to getting one of bytecode_compute, bytecode_fragment, bytecode_tesselation_control, bytecode_tesselation_evaluation, bytecode_vertex.

String **get_stage_compile_error**(stage: ShaderStage) const 🔗

Returns the compilation error message for the given shader `stage`. Equivalent to getting one of compile_error_compute, compile_error_fragment, compile_error_tesselation_control, compile_error_tesselation_evaluation, compile_error_vertex.

---

void **set_stage_bytecode**(stage: ShaderStage, bytecode: PackedByteArray) 🔗

Sets the SPIR-V `bytecode` for the given shader `stage`. Equivalent to setting one of bytecode_compute, bytecode_fragment, bytecode_tesselation_control, bytecode_tesselation_evaluation, bytecode_vertex.

---

void **set_stage_compile_error**(stage: ShaderStage, compile_error: String) 🔗

Sets the compilation error message for the given shader `stage` to `compile_error`. Equivalent to setting one of compile_error_compute, compile_error_fragment, compile_error_tesselation_control, compile_error_tesselation_evaluation, compile_error_vertex.

RectangleShape2D

Inherits: Shape2D **<** Resource **<** RefCounted **<** Object

A 2D rectangle shape used for physics collision.

Description

A 2D rectangle shape, intended for use in physics. Usually used to provide a shape for a CollisionShape2D.

Performance: RectangleShape2D is fast to check collisions against. It is faster than CapsuleShape2D, but slower than CircleShape2D.

Tutorials

- 2D Pong Demo [https://godotengine.org/asset-library/asset/2728]
- 2D Kinematic Character Demo [https://godotengine.org/asset-library/asset/2719]

Properties

Vector2	size	`Vector2(20, 20)`

Property Descriptions

Vector2 **size** = `Vector2(20, 20)` 🔗

- void **set_size**(value: Vector2)
- Vector2 **get_size**()

The rectangle's width and height.

RibbonTrailMesh

Inherits: PrimitiveMesh **<** Mesh **<** Resource **<** RefCounted **<** Object

Represents a straight ribbon-shaped PrimitiveMesh with variable width.

Description

RibbonTrailMesh represents a straight ribbon-shaped mesh with variable width. The ribbon is composed of a number of flat or cross-shaped sections, each with the same section_length and number of section_segments. A curve is sampled along the total length of the ribbon, meaning that the curve determines the size of the ribbon along its length.

This primitive mesh is usually used for particle trails.

Tutorials

- 3D Particle trails
- Particle systems (3D)

Properties

Curve	curve	
float	section_length	`0.2`

int	section_segments	`3`
int	sections	`5`
Shape	shape	`1`
float	size	`1.0`

Enumerations

enum **Shape**: 🔗

Shape **SHAPE_FLAT** = `0`

Gives the mesh a single flat face.

Shape **SHAPE_CROSS** = `1`

Gives the mesh two perpendicular flat faces, making a cross shape.

Property Descriptions

Curve **curve** 🔗

- void **set_curve**(value: Curve)
- Curve **get_curve**()

Determines the size of the ribbon along its length. The size of a particular section segment is obtained by multiplying the baseline size by the value of this curve at the given distance. For values smaller than `0`, the faces will be inverted.

---

float **section_length** = `0.2` 🔗

- void **set_section_length**(value: float)
- float **get_section_length**()

The length of a section of the ribbon.

---

int **section_segments** = `3` 🔗

- void **set_section_segments**(value: int)
- int **get_section_segments**()

The number of segments in a section. The curve is sampled on each segment to determine its size. Higher values result in a more detailed ribbon at the cost of performance.

---

int **sections** = `5` 🔗

- void **set_sections**(value: int)
- int **get_sections**()

The total number of sections on the ribbon.

Shape **shape** = `1` 🔗

- void **set_shape**(value: Shape)
- Shape **get_shape**()

Determines the shape of the ribbon.

---

float **size** = `1.0`

- void **set_size**(value: float)
- float **get_size**()

The baseline size of the ribbon. The size of a particular section segment is obtained by multiplying this size by the value of the curve at the given distance.

RichTextEffect

Inherits: Resource **<** RefCounted **<** Object

A custom effect for a RichTextLabel.

Description

A custom effect for a RichTextLabel, which can be loaded in the RichTextLabel inspector or using RichTextLabel.install_effect.

Note: For a **RichTextEffect** to be usable, a BBCode tag must be defined as a member variable called `bbcode` in the script.

GDScript

```
# The RichTextEffect will be usable like this: `[example]Some
text[/example]`
var bbcode = "example"
```

C#

```
// The RichTextEffect will be usable like this: `[example]Some
text[/example]`
string bbcode = "example";
```

Note: As soon as a RichTextLabel contains at least one **RichTextEffect**, it will continuously process the effect unless the project is paused. This may impact battery life negatively.

Tutorials

- BBCode in RichTextLabel

- RichTextEffect test project (third-party) [https://github.com/Eoin-ONeill-Yokai/Godot-Rich-Text-Effect-Test-Project]

Methods

bool	_process_custom_fx(char_fx: CharFXTransform) virtual const

---

Method Descriptions

bool **_process_custom_fx**(char_fx: CharFXTransform) virtual const

Override this method to modify properties in `char_fx`. The method must return `true` if the character could be transformed successfully. If the method returns `false`, it will skip transformation to avoid displaying broken text.

SceneReplicationConfig

Inherits: Resource < RefCounted < Object

Configuration for properties to synchronize with a MultiplayerSynchronizer.

Methods

void	add_property(path: NodePath, index: int = -1)
Array[NodePath]	get_properties() const
bool	has_property(path: NodePath) const
int	property_get_index(path: NodePath) const
ReplicationMode	property_get_replication_mode(path: NodePath)
bool	property_get_spawn(path: NodePath)
bool	property_get_sync(path: NodePath)
bool	property_get_watch(path: NodePath)

void	property_set_replication_mode(path: NodePath, mode: ReplicationMode)
void	property_set_spawn(path: NodePath, enabled: bool)
void	property_set_sync(path: NodePath, enabled: bool)
void	property_set_watch(path: NodePath, enabled: bool)
void	remove_property(path: NodePath)

Enumerations

enum **ReplicationMode**: 🔗

ReplicationMode **REPLICATION_MODE_NEVER** = `0`

Do not keep the given property synchronized.

ReplicationMode **REPLICATION_MODE_ALWAYS** = `1`

Replicate the given property on process by constantly sending updates using unreliable transfer mode.

ReplicationMode **REPLICATION_MODE_ON_CHANGE** = `2`

Replicate the given property on process by sending updates using reliable transfer mode when its value changes.

Method Descriptions

void **add_property**(path: NodePath, index: int = -1) 🔗

Adds the property identified by the given `path` to the list of the properties being synchronized, optionally passing an `index`.

Note: For details on restrictions and limitations on property synchronization, see MultiplayerSynchronizer.

Array[NodePath] **get_properties**() const 🔗

Returns a list of synchronized property NodePaths.

---

bool **has_property**(path: NodePath) const 🔗

Returns `true` if the given `path` is configured for synchronization.

---

int **property_get_index**(path: NodePath) const 🔗

Finds the index of the given `path`.

ReplicationMode **property_get_replication_mode**(path: NodePath) 🔗

Returns the replication mode for the property identified by the given `path`. See ReplicationMode.

---

bool **property_get_spawn**(path: NodePath)

Returns `true` if the property identified by the given `path` is configured to be synchronized on spawn.

bool **property_get_sync**(path: NodePath)

Deprecated: Use property_get_replication_mode instead.

Returns `true` if the property identified by the given `path` is configured to be synchronized on process.

bool **property_get_watch**(path: NodePath)

Deprecated: Use property_get_replication_mode instead.

Returns `true` if the property identified by the given `path` is configured to be reliably synchronized when changes are detected on process.

void **property_set_replication_mode**(path: NodePath, mode: ReplicationMode)

Sets the synchronization mode for the property identified by the given `path`. See ReplicationMode.

void **property_set_spawn**(path: NodePath, enabled: bool)

Sets whether the property identified by the given `path` is configured to be synchronized on spawn.

void **property_set_sync**(path: NodePath, enabled: bool)

Deprecated: Use property_set_replication_mode with REPLICATION_MODE_ALWAYS instead.

Sets whether the property identified by the given `path` is configured to be synchronized on process.

void **property_set_watch**(path: NodePath, enabled: bool) 🔗

Deprecated: Use property_set_replication_mode with REPLICATION_MODE_ON_CHANGE instead.

Sets whether the property identified by the given `path` is configured to be reliably synchronized when changes are detected on process.

void **remove_property**(path: NodePath) 🔗

Removes the property identified by the given `path` from the configuration.

Script

Inherits: Resource **<** RefCounted **<** Object

Inherited By: CSharpScript, GDScript, ScriptExtension

A class stored as a resource.

Description

A class stored as a resource. A script extends the functionality of all objects that instantiate it.

This is the base class for all scripts and should not be used directly. Trying to create a new script with this class will result in an error.

The `new` method of a script subclass creates a new instance. Object.set_script extends an existing object, if that object's class matches one of the script's base classes.

Tutorials

- Scripting documentation index

Properties

String	source_code

Methods

bool	can_instantiate() const
Script	get_base_script() const
StringName	get_global_name() const
StringName	get_instance_base_type() const
Variant	get_property_default_value(property: StringName)
Dictionary	get_script_constant_map()
Array[Dictionary]	get_script_method_list()
Array[Dictionary]	get_script_property_list()
Array[Dictionary]	get_script_signal_list()
bool	has_script_signal(signal_name: StringName) const
bool	has_source_code() const

bool	instance_has(base_object: Object) const
bool	is_abstract() const
bool	is_tool() const
Error	reload(keep_state: bool = false)

Property Descriptions

String **source_code** 🔗

- void **set_source_code**(value: String)
- String **get_source_code**()

The script source code or an empty string if source code is not available. When set, does not reload the class implementation automatically.

Method Descriptions

bool **can_instantiate**() const 🔗

Returns `true` if the script can be instantiated.

Script **get_base_script**() const 🔗

Returns the script directly inherited by this script.

StringName **get_global_name**() const 🔗

Returns the class name associated with the script, if there is one. Returns an empty string otherwise.

To give the script a global name, you can use the `class_name` keyword in GDScript and the `[GlobalClass]` attribute in C#.

GDScript

```
class_name MyNode
extends Node
```

C#

```
using Godot;

[GlobalClass]
public partial class MyNode : Node
{
}
```

StringName **get_instance_base_type**() const 🔗

Returns the script's base type.

Variant **get_property_default_value**(property: StringName) 🔗

Returns the default value of the specified property.

Dictionary **get_script_constant_map**() 🔗

Returns a dictionary containing constant names and their values.

Array[Dictionary] **get_script_method_list**()

Returns the list of methods in this **Script**.

Array[Dictionary] **get_script_property_list**()

Returns the list of properties in this **Script**.

Array[Dictionary] **get_script_signal_list**()

Returns the list of user signals defined in this **Script**.

---

bool **has_script_signal**(signal_name: StringName) const

Returns `true` if the script, or a base class, defines a signal with the given name.

---

bool **has_source_code**() const

Returns `true` if the script contains non-empty source code.

Note: If a script does not have source code, this does not mean that it is invalid or unusable. For example, a GDScript that was exported with binary tokenization has no source code, but still behaves as expected and could be instantiated. This can be checked with can_instantiate.

---

bool **instance_has**(base_object: Object) const

Returns `true` if `base_object` is an instance of this script.

---

bool **is_abstract**() const 🔗

Returns `true` if the script is an abstract script. An abstract script does not have a constructor and cannot be instantiated.

---

bool **is_tool**() const 🔗

Returns `true` if the script is a tool script. A tool script can run in the editor.

Error **reload**(keep_state: bool = false) 🔗

Reloads the script's class implementation. Returns an error code.

ScriptExtension

Inherits: Script **<** Resource **<** RefCounted **<** Object

There is currently no description for this class. Please help us by contributing one!

Methods

bool	_can_instantiate() virtual const
bool	_editor_can_reload_from_file() virtual
Script	_get_base_script() virtual const
String	_get_class_icon_path() virtual const
Dictionary	_get_constants() virtual const
Array[Dictionary]	_get_documentation() virtual const
StringName	_get_global_name() virtual const
StringName	_get_instance_base_type() virtual const

ScriptLanguage	_get_language() virtual const
int	_get_member_line(member: StringName) virtual const
Array[StringName]	_get_members() virtual const
Dictionary	_get_method_info(method: StringName) virtual const
Variant	_get_property_default_value(property: StringName) virtual const
Variant	_get_rpc_config() virtual const
Variant	_get_script_method_argument_count(method: StringName) virtual const
Array[Dictionary]	_get_script_method_list() virtual const
Array[Dictionary]	_get_script_property_list() virtual const
Array[Dictionary]	_get_script_signal_list() virtual const
String	_get_source_code() virtual const

bool	_has_method(method: StringName) virtual const
bool	_has_property_default_value(property: StringName) virtual const
bool	_has_script_signal(signal: StringName) virtual const
bool	_has_source_code() virtual const
bool	_has_static_method(method: StringName) virtual const
bool	_inherits_script(script: Script) virtual const
`void*`	_instance_create(for_object: Object) virtual const
bool	_instance_has(object: Object) virtual const
bool	_is_abstract() virtual const
bool	_is_placeholder_fallback_enabled() virtual const
bool	_is_tool() virtual const

bool	_is_valid() virtual const
void	_placeholder_erased(placeholder: `void*`) virtual
`void*`	_placeholder_instance_create(for_object: Object) virtual const
Error	_reload(keep_state: bool) virtual
void	_set_source_code(code: String) virtual
void	_update_exports() virtual

Method Descriptions

bool **_can_instantiate**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

bool **_editor_can_reload_from_file**() virtual 🔗

There is currently no description for this method. Please help us by contributing one!

Script **_get_base_script**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

String **_get_class_icon_path**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

Dictionary **_get_constants**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

Array[Dictionary] **_get_documentation**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

StringName **_get_global_name**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

StringName **_get_instance_base_type**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

ScriptLanguage **_get_language**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

int **_get_member_line**(member: StringName) virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

Array[StringName] **_get_members**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

Dictionary **_get_method_info**(method: StringName) virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

Variant **_get_property_default_value**(property: StringName) virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

Variant **_get_rpc_config**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

Variant **_get_script_method_argument_count**(method: StringName) virtual const 🔗

Return the expected argument count for the given `method`, or `null` if it can't be determined (which will then fall back to the default behavior).

Array[Dictionary] **_get_script_method_list**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

Array[Dictionary] **_get_script_property_list**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

Array[Dictionary] **_get_script_signal_list**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

String **_get_source_code**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

bool **_has_method**(method: StringName) virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

bool **_has_property_default_value**(property: StringName) virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

bool **_has_script_signal**(signal: StringName) virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

bool **_has_source_code**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

bool **_has_static_method**(method: StringName) virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

bool **_inherits_script**(script: Script) virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

`void*` **_instance_create**(for_object: Object) virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

bool **_instance_has**(object: Object) virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

bool **_is_abstract**() virtual const 🔗

Returns `true` if the script is an abstract script. An abstract script does not have a constructor and cannot be instantiated.

---

bool **_is_placeholder_fallback_enabled**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

bool **_is_tool**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

bool **_is_valid**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

void **_placeholder_erased**(placeholder: `void*`) virtual 🔗

There is currently no description for this method. Please help us by contributing one!

`void*` **_placeholder_instance_create**(for_object: Object) virtual const

There is currently no description for this method. Please help us by contributing one!

Error **_reload**(keep_state: bool) virtual

There is currently no description for this method. Please help us by contributing one!

---

void **_set_source_code**(code: String) virtual

There is currently no description for this method. Please help us by contributing one!

---

void **_update_exports**() virtual

There is currently no description for this method. Please help us by contributing one!

SegmentShape2D

Inherits: Shape2D **<** Resource **<** RefCounted **<** Object

A 2D line segment shape used for physics collision.

Description

A 2D line segment shape, intended for use in physics. Usually used to provide a shape for a CollisionShape2D.

Properties

Vector2	a	`Vector2(0, 0)`
Vector2	b	`Vector2(0, 10)`

Property Descriptions

Vector2 **a** = `Vector2(0, 0)`

- void **set_a**(value: Vector2)
- Vector2 **get_a**()

The segment's first point position.

Vector2 **b** = `Vector2(0, 10)`

- void **set_b**(value: Vector2)
- Vector2 **get_b**()

The segment's second point position.

SeparationRayShape2D

Inherits: Shape2D **<** Resource **<** RefCounted **<** Object

A 2D ray shape used for physics collision that tries to separate itself from any collider.

Description

A 2D ray shape, intended for use in physics. Usually used to provide a shape for a CollisionShape2D. When a **SeparationRayShape2D** collides with an object, it tries to separate itself from it by moving its endpoint to the collision point. For example, a **SeparationRayShape2D** next to a character can allow it to instantly move up when touching stairs.

Properties

float	length	`20.0`
bool	slide_on_slope	`false`

Property Descriptions

float **length** = `20.0` 🔗

- void **set_length**(value: float)

- float **get_length**()

The ray's length.

---

float **slide_on_slope** = `false` 🔗

- void **set_slide_on_slope**(value: bool)
- bool **get_slide_on_slope**()

If `false` (default), the shape always separates and returns a normal along its own direction.

If `true`, the shape can return the correct normal and separate in any direction, allowing sliding motion on slopes.

SeparationRayShape3D

Inherits: Shape3D **<** Resource **<** RefCounted **<** Object

A 3D ray shape used for physics collision that tries to separate itself from any collider.

Description

A 3D ray shape, intended for use in physics. Usually used to provide a shape for a CollisionShape3D. When a **SeparationRayShape3D** collides with an object, it tries to separate itself from it by moving its endpoint to the collision point. For example, a **SeparationRayShape3D** next to a character can allow it to instantly move up when touching stairs.

Properties

float	length	`1.0`
bool	slide_on_slope	`false`

Property Descriptions

float **length** = `1.0` 🔗

- void **set_length**(value: float)

- float **get_length**()

The ray's length.

---

bool **slide_on_slope** = `false`

- void **set_slide_on_slope**(value: bool)
- bool **get_slide_on_slope**()

If `false` (default), the shape always separates and returns a normal along its own direction.

If `true`, the shape can return the correct normal and separate in any direction, allowing sliding motion on slopes.

Shader

Inherits: Resource **<** RefCounted **<** Object

Inherited By: VisualShader

A shader implemented in the Godot shading language.

Description

A custom shader program implemented in the Godot shading language, saved with the `.gdshader` extension.

This class is used by a ShaderMaterial and allows you to write your own custom behavior for rendering visual items or updating particle information. For a detailed explanation and usage, please see the tutorials linked below.

Tutorials

- Shaders documentation index

Properties

String	code	`""`

Methods

Texture2D	get_default_texture_parameter(name: StringName, index: int = 0) const
Mode	get_mode() const
Array	get_shader_uniform_list(get_groups: bool = false)
void	set_default_texture_parameter(name: StringName, texture: Texture2D, index: int = 0)

Enumerations

enum **Mode**: 🔗

Mode **MODE_SPATIAL** = `0`

Mode used to draw all 3D objects.

Mode **MODE_CANVAS_ITEM** = `1`

Mode used to draw all 2D objects.

Mode **MODE_PARTICLES** = `2`

Mode used to calculate particle information on a per-particle basis. Not used for drawing.

Mode **MODE_SKY** = `3`

Mode used for drawing skies. Only works with shaders attached to Sky objects.

Mode **MODE_FOG** = `4`

Mode used for setting the color and density of volumetric fog effect.

Property Descriptions

String **code** = `""`

- void **set_code**(value: String)
- String **get_code**()

Returns the shader's code as the user has written it, not the full generated code used internally.

Method Descriptions

Texture2D **get_default_texture_parameter**(name: StringName, index: int = 0) const

Returns the texture that is set as default for the specified parameter.

Note: `name` must match the name of the uniform in the code exactly.

Note: If the sampler array is used use `index` to access the specified texture.

Mode **get_mode**() const

Returns the shader mode for the shader.

Array **get_shader_uniform_list**(get_groups: bool = false)

Get the list of shader uniforms that can be assigned to a ShaderMaterial, for use with ShaderMaterial.set_shader_parameter and ShaderMaterial.get_shader_parameter. The parameters returned are contained in dictionaries in a similar format to the ones returned by Object.get_property_list.

If argument `get_groups` is true, parameter grouping hints will be provided.

---

void **set_default_texture_parameter**(name: StringName, texture: Texture2D, index: int = 0) 🔗

Sets the default texture to be used with a texture uniform. The default is used if a texture is not set in the ShaderMaterial.

Note: `name` must match the name of the uniform in the code exactly.

Note: If the sampler array is used use `index` to access the specified texture.

ShaderInclude

Inherits: Resource **<** RefCounted **<** Object

A snippet of shader code to be included in a Shader with `#include`.

Description

A shader include file, saved with the `.gdshaderinc` extension. This class allows you to define a custom shader snippet that can be included in a Shader by using the preprocessor directive `#include`, followed by the file path (e.g. `#include "res://shader_lib.gdshaderinc"`). The snippet doesn't have to be a valid shader on its own.

Tutorials

- Shader preprocessor

Properties

String	code	`""`

Property Descriptions

String **code** = `""` 🔗

- void **set_code**(value: String)
- String **get_code**()

Returns the code of the shader include file. The returned text is what the user has written, not the full generated code used internally.

ShaderMaterial

Inherits: Material **<** Resource **<** RefCounted **<** Object

A material defined by a custom Shader program and the values of its shader parameters.

Description

A material that uses a custom Shader program to render visual items (canvas items, meshes, skies, fog), or to process particles. Compared to other materials, **ShaderMaterial** gives deeper control over the generated shader code. For more information, see the shaders documentation index below.

Multiple **ShaderMaterial**s can use the same shader and configure different values for the shader uniforms.

Note: For performance reasons, the Resource.changed signal is only emitted when the Resource.resource_name changes. Only in editor, it is also emitted for shader changes.

Tutorials

- Shaders documentation index

Properties

Shader	shader

Methods

Variant	get_shader_parameter(param: StringName) const
void	set_shader_parameter(param: StringName, value: Variant)

Property Descriptions

Shader **shader** 🔗

- void **set_shader**(value: Shader)
- Shader **get_shader**()

The Shader program used to render this material.

Method Descriptions

Variant **get_shader_parameter**(param: StringName) const 🔗

Returns the current value set for this material of a uniform in the shader.

---

void **set_shader_parameter**(param: StringName, value: Variant) 🔗

Changes the value set for this material of a uniform in the shader.

Note: `param` is case-sensitive and must match the name of the uniform in the code exactly (not the capitalized name in the inspector).

Note: Changes to the shader uniform will be effective on all instances using this **ShaderMaterial**. To prevent this, use per-instance uniforms with GeometryInstance3D.set_instance_shader_parameter or duplicate the **ShaderMaterial** resource using Resource.duplicate. Per-instance uniforms allow for better shader reuse and are therefore faster, so they should be preferred over duplicating the **ShaderMaterial** when possible.

Shape2D

Inherits: Resource < RefCounted < Object

Inherited By: CapsuleShape2D, CircleShape2D, ConcavePolygonShape2D, ConvexPolygonShape2D, RectangleShape2D, SegmentShape2D, SeparationRayShape2D, WorldBoundaryShape2D

Abstract base class for 2D shapes used for physics collision.

Description

Abstract base class for all 2D shapes, intended for use in physics.

Performance: Primitive shapes, especially CircleShape2D, are fast to check collisions against. ConvexPolygonShape2D is slower, and ConcavePolygonShape2D is the slowest.

Tutorials

- Physics introduction

Properties

float	custom_solver_bias	`0.0`

Methods

bool	collide(local_xform: Transform2D, with_shape: Shape2D, shape_xform: Transform2D)

PackedVector2Array	collide_and_get_contacts(local_xform: Transform2D, with_shape: Shape2D, shape_xform: Transform2D)
bool	collide_with_motion(local_xform: Transform2D, local_motion: Vector2, with_shape: Shape2D, shape_xform: Transform2D, shape_motion: Vector2)
PackedVector2Array	collide_with_motion_and_get_contacts(local_xform: Transform2D, local_motion: Vector2, with_shape: Shape2D, shape_xform: Transform2D, shape_motion: Vector2)
void	draw(canvas_item: RID, color: Color)
Rect2	get_rect() const

Property Descriptions

float **custom_solver_bias** = `0.0` 🔗

- void **set_custom_solver_bias**(value: float)
- float **get_custom_solver_bias**()

The shape's custom solver bias. Defines how much bodies react to enforce contact separation when this shape is involved.

When set to `0`, the default value from ProjectSettings.physics/2d/solver/default_contact_bias is used.

Method Descriptions

bool **collide**(local_xform: Transform2D, with_shape: Shape2D, shape_xform: Transform2D) 🔗

Returns `true` if this shape is colliding with another.

This method needs the transformation matrix for this shape (`local_xform`), the shape to check collisions with (`with_shape`), and the transformation matrix of that shape (`shape_xform`).

PackedVector2Array **collide_and_get_contacts**(local_xform: Transform2D, with_shape: Shape2D, shape_xform: Transform2D) 🔗

Returns a list of contact point pairs where this shape touches another.

If there are no collisions, the returned list is empty. Otherwise, the returned list contains contact points arranged in pairs, with entries alternating between points on the boundary of this shape and points on the boundary of `with_shape`.

A collision pair A, B can be used to calculate the collision normal with `(B - A).normalized()`, and the collision depth with `(B - A).length()`. This information is typically used to separate shapes, particularly in collision solvers.

This method needs the transformation matrix for this shape (`local_xform`), the shape to check collisions with (`with_shape`), and the transformation matrix of that shape (`shape_xform`).

---

bool **collide_with_motion**(local_xform: Transform2D, local_motion: Vector2, with_shape: Shape2D, shape_xform: Transform2D, shape_motion: Vector2) 🔗

Returns whether this shape would collide with another, if a given movement was applied.

This method needs the transformation matrix for this shape (`local_xform`), the movement to test on this shape (`local_motion`), the shape to check collisions with (`with_shape`), the transformation matrix of that shape (`shape_xform`), and the movement to test onto the other object (`shape_motion`).

PackedVector2Array **collide_with_motion_and_get_contacts**(local_xform: Transform2D, local_motion: Vector2, with_shape: Shape2D, shape_xform: Transform2D, shape_motion: Vector2) 🔗

Returns a list of contact point pairs where this shape would touch another, if a given movement was applied.

If there would be no collisions, the returned list is empty. Otherwise, the returned list contains contact points arranged in pairs, with entries alternating between points on the boundary of this shape and points on the boundary of `with_shape`.

A collision pair A, B can be used to calculate the collision normal with `(B - A).normalized()`, and the collision depth with `(B - A).length()`. This information is typically used to separate shapes, particularly in collision solvers.

This method needs the transformation matrix for this shape (`local_xform`), the movement to test on this shape (`local_motion`), the shape to check collisions with (`with_shape`), the transformation matrix of that shape (`shape_xform`), and the movement to test onto the other object (`shape_motion`).

---

void **draw**(canvas_item: RID, color: Color) 🔗

Draws a solid shape onto a CanvasItem with the RenderingServer API filled with the specified `color`. The exact drawing method is specific for each shape and cannot be configured.

Rect2 **get_rect**() const

Returns a Rect2 representing the shapes boundary.

Shape3D

Inherits: Resource **<** RefCounted **<** Object

Inherited By: BoxShape3D, CapsuleShape3D, ConcavePolygonShape3D, ConvexPolygonShape3D, CylinderShape3D, HeightMapShape3D, SeparationRayShape3D, SphereShape3D, WorldBoundaryShape3D

Abstract base class for 3D shapes used for physics collision.

Description

Abstract base class for all 3D shapes, intended for use in physics.

Performance: Primitive shapes, especially SphereShape3D, are fast to check collisions against. ConvexPolygonShape3D and HeightMapShape3D are slower, and ConcavePolygonShape3D is the slowest.

Tutorials

- Physics introduction

Properties

float	custom_solver_bias	`0.0`
float	margin	`0.04`

Methods

ArrayMesh	get_debug_mesh()

Property Descriptions

float **custom_solver_bias** = `0.0` 🔗

- void **set_custom_solver_bias**(value: float)
- float **get_custom_solver_bias**()

The shape's custom solver bias. Defines how much bodies react to enforce contact separation when this shape is involved.

When set to `0`, the default value from ProjectSettings.physics/3d/solver/default_contact_bias is used.

---

float **margin** = `0.04` 🔗

- void **set_margin**(value: float)
- float **get_margin**()

The collision margin for the shape. This is not used in Godot Physics.

Collision margins allow collision detection to be more efficient by adding an extra shell around shapes. Collision algorithms are more expensive when objects overlap by more than their margin, so a higher value for margins is better for performance, at the cost of accuracy around edges as it makes them less sharp.

Method Descriptions

ArrayMesh **get_debug_mesh**()

Returns the ArrayMesh used to draw the debug collision for this **Shape3D**.

Shortcut

Inherits: Resource **<** RefCounted **<** Object

A shortcut for binding input.

Description

Shortcuts are commonly used for interacting with a Control element from an InputEvent (also known as hotkeys).

One shortcut can contain multiple InputEvent's, allowing the possibility of triggering one action with multiple different inputs.

Properties

Array	events	[ ]

Methods

String	get_as_text() const
bool	has_valid_event() const
bool	matches_event(event: InputEvent) const

Property Descriptions

Array **events** = `[]`

- void **set_events**(value: Array)
- Array **get_events**()

The shortcut's InputEvent array.

Generally the InputEvent used is an InputEventKey, though it can be any InputEvent, including an InputEventAction.

Method Descriptions

String **get_as_text**() const

Returns the shortcut's first valid InputEvent as a String.

---

bool **has_valid_event**() const

Returns whether events contains an InputEvent which is valid.

---

bool **matches_event**(event: InputEvent) const

Returns whether any InputEvent in events equals `event`.

SkeletonModification2D

Experimental: This class may be changed or removed in future versions.

Inherits: Resource **<** RefCounted **<** Object

Inherited By: SkeletonModification2DCCDIK, SkeletonModification2DFABRIK, SkeletonModification2DJiggle, SkeletonModification2DLookAt, SkeletonModification2DPhysicalBones, SkeletonModification2DStackHolder, SkeletonModification2DTwoBoneIK

Base class for resources that operate on Bone2Ds in a Skeleton2D.

Description

This resource provides an interface that can be expanded so code that operates on Bone2D nodes in a Skeleton2D can be mixed and matched together to create complex interactions.

This is used to provide Godot with a flexible and powerful Inverse Kinematics solution that can be adapted for many different uses.

Properties

bool	enabled	`true`
int	execution_mode	`0`

Methods

void	_draw_editor_gizmo() virtual
void	_execute(delta: float) virtual
void	_setup_modification(modification_stack: SkeletonModificationStack2D) virtual
float	clamp_angle(angle: float, min: float, max: float, invert: bool)
bool	get_editor_draw_gizmo() const
bool	get_is_setup() const
SkeletonModificationStack2D	get_modification_stack()
void	set_editor_draw_gizmo(draw_gizmo: bool)
void	set_is_setup(is_setup: bool)

Property Descriptions

bool **enabled** = `true` 🔗

- void **set_enabled**(value: bool)
- bool **get_enabled**()

If `true`, the modification's _execute function will be called by the SkeletonModificationStack2D.

---

int **execution_mode** = `0`

- void **set_execution_mode**(value: int)
- int **get_execution_mode**()

The execution mode for the modification. This tells the modification stack when to execute the modification. Some modifications have settings that are only available in certain execution modes.

Method Descriptions

void **_draw_editor_gizmo**() virtual

Used for drawing **editor-only** modification gizmos. This function will only be called in the Godot editor and can be overridden to draw custom gizmos.

Note: You will need to use the Skeleton2D from SkeletonModificationStack2D.get_skeleton and it's draw functions, as the **SkeletonModification2D** resource cannot draw on its own.

---

void **_execute**(delta: float) virtual

Executes the given modification. This is where the modification performs whatever function it is designed to do.

---

void **_setup_modification**(modification_stack: SkeletonModificationStack2D) virtual

Called when the modification is setup. This is where the modification performs initialization.

float **clamp_angle**(angle: float, min: float, max: float, invert: bool)

Takes an angle and clamps it so it is within the passed-in `min` and `max` range. `invert` will inversely clamp the angle, clamping it to the range outside of the given bounds.

bool **get_editor_draw_gizmo**() const

Returns whether this modification will call _draw_editor_gizmo in the Godot editor to draw modification-specific gizmos.

bool **get_is_setup**() const

Returns whether this modification has been successfully setup or not.

SkeletonModificationStack2D **get_modification_stack**()

Returns the SkeletonModificationStack2D that this modification is bound to. Through the modification stack, you can access the Skeleton2D the modification is operating on.

void **set_editor_draw_gizmo**(draw_gizmo: bool)

Sets whether this modification will call _draw_editor_gizmo in the Godot editor to draw modification-specific gizmos.

void **set_is_setup**(is_setup: bool)

Manually allows you to set the setup state of the modification. This function should only rarely be used, as the SkeletonModificationStack2D the modification is bound to should handle setting the modification up.

SkeletonModification2DCCDIK

Experimental: This class may be changed or removed in future versions.

Inherits: SkeletonModification2D **<** Resource **<** RefCounted **<** Object

A modification that uses CCDIK to manipulate a series of bones to reach a target in 2D.

Description

This SkeletonModification2D uses an algorithm called Cyclic Coordinate Descent Inverse Kinematics, or CCDIK, to manipulate a chain of bones in a Skeleton2D so it reaches a defined target.

CCDIK works by rotating a set of bones, typically called a "bone chain", on a single axis. Each bone is rotated to face the target from the tip (by default), which over a chain of bones allow it to rotate properly to reach the target. Because the bones only rotate on a single axis, CCDIK *can* look more robotic than other IK solvers.

Note: The CCDIK modifier has `ccdik_joints`, which are the data objects that hold the data for each joint in the CCDIK chain. This is different from a bone! CCDIK joints hold the data needed for each bone in the bone chain used by CCDIK.

CCDIK also fully supports angle constraints, allowing for more control over how a solution is met.

Properties

int	ccdik_data_chain_length	`0`
NodePath	target_nodepath	`NodePath("")`
NodePath	tip_nodepath	`NodePath("")`

Methods

NodePath	get_ccdik_joint_bone2d_node(joint_idx: int) const
int	get_ccdik_joint_bone_index(joint_idx: int) const
bool	get_ccdik_joint_constraint_angle_invert(joint_idx: int) const
float	get_ccdik_joint_constraint_angle_max(joint_idx: int) const
float	get_ccdik_joint_constraint_angle_min(joint_idx: int) const
bool	get_ccdik_joint_enable_constraint(joint_idx: int) const

bool	get_ccdik_joint_rotate_from_joint(joint_idx: int) const
void	set_ccdik_joint_bone2d_node(joint_idx: int, bone2d_nodepath: NodePath)
void	set_ccdik_joint_bone_index(joint_idx: int, bone_idx: int)
void	set_ccdik_joint_constraint_angle_invert(joint_idx: int, invert: bool)
void	set_ccdik_joint_constraint_angle_max(joint_idx: int, angle_max: float)
void	set_ccdik_joint_constraint_angle_min(joint_idx: int, angle_min: float)
void	set_ccdik_joint_enable_constraint(joint_idx: int, enable_constraint: bool)
void	set_ccdik_joint_rotate_from_joint(joint_idx: int, rotate_from_joint: bool)

Property Descriptions

int **ccdik_data_chain_length** = 0 🔗

- void **set_ccdik_data_chain_length**(value: int)

- int **get_ccdik_data_chain_length**()

The number of CCDIK joints in the CCDIK modification.

NodePath **target_nodepath** = `NodePath("")`

- void **set_target_node**(value: NodePath)
- NodePath **get_target_node**()

The NodePath to the node that is the target for the CCDIK modification. This node is what the CCDIK chain will attempt to rotate the bone chain to.

NodePath **tip_nodepath** = `NodePath("")`

- void **set_tip_node**(value: NodePath)
- NodePath **get_tip_node**()

The end position of the CCDIK chain. Typically, this should be a child of a Bone2D node attached to the final Bone2D in the CCDIK chain.

Method Descriptions

NodePath **get_ccdik_joint_bone2d_node**(joint_idx: int) const

Returns the Bone2D node assigned to the CCDIK joint at `joint_idx`.

int **get_ccdik_joint_bone_index**(joint_idx: int) const

Returns the index of the Bone2D node assigned to the CCDIK joint at `joint_idx`.

---

bool **get_ccdik_joint_constraint_angle_invert**(joint_idx: int) const 🔗

Returns whether the CCDIK joint at `joint_idx` uses an inverted joint constraint. See set_ccdik_joint_constraint_angle_invert for details.

---

float **get_ccdik_joint_constraint_angle_max**(joint_idx: int) const 🔗

Returns the maximum angle constraint for the joint at `joint_idx`.

---

float **get_ccdik_joint_constraint_angle_min**(joint_idx: int) const 🔗

Returns the minimum angle constraint for the joint at `joint_idx`.

---

bool **get_ccdik_joint_enable_constraint**(joint_idx: int) const 🔗

Returns whether angle constraints on the CCDIK joint at `joint_idx` are enabled.

---

bool **get_ccdik_joint_rotate_from_joint**(joint_idx: int) const 🔗

Returns whether the joint at `joint_idx` is set to rotate from the joint, `true`, or to rotate from the tip, `false`. The default is to rotate from the tip.

---

void **set_ccdik_joint_bone2d_node**(joint_idx: int, bone2d_nodepath: NodePath) 🔗

Sets the Bone2D node assigned to the CCDIK joint at `joint_idx`.

---

void **set_ccdik_joint_bone_index**(joint_idx: int, bone_idx: int)

Sets the bone index, `bone_idx`, of the CCDIK joint at `joint_idx`. When possible, this will also update the `bone2d_node` of the CCDIK joint based on data provided by the linked skeleton.

---

void **set_ccdik_joint_constraint_angle_invert**(joint_idx: int, invert: bool)

Sets whether the CCDIK joint at `joint_idx` uses an inverted joint constraint.

An inverted joint constraint only constraints the CCDIK joint to the angles *outside of* the inputted minimum and maximum angles. For this reason, it is referred to as an inverted joint constraint, as it constraints the joint to the outside of the inputted values.

---

void **set_ccdik_joint_constraint_angle_max**(joint_idx: int, angle_max: float)

Sets the maximum angle constraint for the joint at `joint_idx`.

---

void **set_ccdik_joint_constraint_angle_min**(joint_idx: int, angle_min: float)

Sets the minimum angle constraint for the joint at `joint_idx`.

---

void **set_ccdik_joint_enable_constraint**(joint_idx: int, enable_constraint: bool) 🔗

Determines whether angle constraints on the CCDIK joint at `joint_idx` are enabled. When `true`, constraints will be enabled and taken into account when solving.

---

void **set_ccdik_joint_rotate_from_joint**(joint_idx: int, rotate_from_joint: bool) 🔗

Sets whether the joint at `joint_idx` is set to rotate from the joint, `true`, or to rotate from the tip, `false`.

SkeletonModification2DFABRIK

Experimental: This class may be changed or removed in future versions.

Inherits: SkeletonModification2D **<** Resource **<** RefCounted **<** Object

A modification that uses FABRIK to manipulate a series of Bone2D nodes to reach a target.

Description

This SkeletonModification2D uses an algorithm called Forward And Backward Reaching Inverse Kinematics, or FABRIK, to rotate a bone chain so that it reaches a target.

FABRIK works by knowing the positions and lengths of a series of bones, typically called a "bone chain". It first starts by running a forward pass, which places the final bone at the target's position. Then all other bones are moved towards the tip bone, so they stay at the defined bone length away. Then a backwards pass is performed, where the root/first bone in the FABRIK chain is placed back at the origin. Then all other bones are moved so they stay at the defined bone length away. This positions the bone chain so that it reaches the target when possible, but all of the bones stay the correct length away from each other.

Because of how FABRIK works, it often gives more natural results than those seen in SkeletonModification2DCCDIK. FABRIK also supports angle constraints, which are fully taken into account when solving.

Note: The FABRIK modifier has `fabrik_joints`, which are the data objects that hold the data for each joint in the FABRIK chain. This is different from Bone2D nodes! FABRIK joints hold the data needed for each Bone2D in the bone chain used by FABRIK.

To help control how the FABRIK joints move, a magnet vector can be passed, which can nudge the bones in a certain direction prior to solving, giving a level of control over the final result.

Properties

int	fabrik_data_chain_length	`0`
NodePath	target_nodepath	`NodePath("")`

Methods

NodePath	get_fabrik_joint_bone2d_node(joint_idx: int) const
int	get_fabrik_joint_bone_index(joint_idx: int) const
Vector2	get_fabrik_joint_magnet_position(joint_idx: int) const
bool	get_fabrik_joint_use_target_rotation(joint_idx: int) const

void	set_fabrik_joint_bone2d_node(joint_idx: int, bone2d_nodepath: NodePath)
void	set_fabrik_joint_bone_index(joint_idx: int, bone_idx: int)
void	set_fabrik_joint_magnet_position(joint_idx: int, magnet_position: Vector2)
void	set_fabrik_joint_use_target_rotation(joint_idx: int, use_target_rotation: bool)

Property Descriptions

int **fabrik_data_chain_length** = `0` 🔗

- void **set_fabrik_data_chain_length**(value: int)
- int **get_fabrik_data_chain_length**()

The number of FABRIK joints in the FABRIK modification.

NodePath **target_nodepath** = `NodePath("")` 🔗

- void **set_target_node**(value: NodePath)
- NodePath **get_target_node**()

The NodePath to the node that is the target for the FABRIK modification. This node is what the FABRIK chain will attempt to rotate the bone chain to.

Method Descriptions

NodePath **get_fabrik_joint_bone2d_node**(joint_idx: int) const 🔗

Returns the Bone2D node assigned to the FABRIK joint at `joint_idx`.

int **get_fabrik_joint_bone_index**(joint_idx: int) const 🔗

Returns the index of the Bone2D node assigned to the FABRIK joint at `joint_idx`.

Vector2 **get_fabrik_joint_magnet_position**(joint_idx: int) const 🔗

Returns the magnet position vector for the joint at `joint_idx`.

bool **get_fabrik_joint_use_target_rotation**(joint_idx: int) const 🔗

Returns whether the joint is using the target's rotation rather than allowing FABRIK to rotate the joint. This option only applies to the tip/final joint in the chain.

void **set_fabrik_joint_bone2d_node**(joint_idx: int, bone2d_nodepath: NodePath) 🔗

Sets the Bone2D node assigned to the FABRIK joint at `joint_idx`.

void **set_fabrik_joint_bone_index**(joint_idx: int, bone_idx: int) 🔗

Sets the bone index, `bone_idx`, of the FABRIK joint at `joint_idx`. When possible, this will also update the `bone2d_node` of the FABRIK joint based on data provided by the linked skeleton.

---

void **set_fabrik_joint_magnet_position**(joint_idx: int, magnet_position: Vector2) 🔗

Sets the magnet position vector for the joint at `joint_idx`.

---

void **set_fabrik_joint_use_target_rotation**(joint_idx: int, use_target_rotation: bool) 🔗

Sets whether the joint at `joint_idx` will use the target node's rotation rather than letting FABRIK rotate the node.

Note: This option only works for the tip/final joint in the chain. For all other nodes, this option will be ignored.

SkeletonModification2DJiggle

Experimental: This class may be changed or removed in future versions.

Inherits: SkeletonModification2D **<** Resource **<** RefCounted **<** Object

A modification that jiggles Bone2D nodes as they move towards a target.

Description

This modification moves a series of bones, typically called a bone chain, towards a target. What makes this modification special is that it calculates the velocity and acceleration for each bone in the bone chain, and runs a very light physics-like calculation using the inputted values. This allows the bones to overshoot the target and "jiggle" around. It can be configured to act more like a spring, or sway around like cloth might.

This modification is useful for adding additional motion to things like hair, the edges of clothing, and more. It has several settings to that allow control over how the joint moves when the target moves.

Note: The Jiggle modifier has `jiggle_joints`, which are the data objects that hold the data for each joint in the Jiggle chain. This is different from than Bone2D nodes! Jiggle joints hold the data needed for each Bone2D in the bone chain used by the Jiggle modification.

Properties

float	damping	`0.75`
Vector2	gravity	`Vector2(0, 6)`
int	jiggle_data_chain_length	`0`
float	mass	`0.75`
float	stiffness	`3.0`
NodePath	target_nodepath	`NodePath("")`
bool	use_gravity	`false`

Methods

int	get_collision_mask() const
NodePath	get_jiggle_joint_bone2d_node(joint_idx: int) const
int	get_jiggle_joint_bone_index(joint_idx: int) const
float	get_jiggle_joint_damping(joint_idx: int) const

Vector2	get_jiggle_joint_gravity(joint_idx: int) const
float	get_jiggle_joint_mass(joint_idx: int) const
bool	get_jiggle_joint_override(joint_idx: int) const
float	get_jiggle_joint_stiffness(joint_idx: int) const
bool	get_jiggle_joint_use_gravity(joint_idx: int) const
bool	get_use_colliders() const
void	set_collision_mask(collision_mask: int)
void	set_jiggle_joint_bone2d_node(joint_idx: int, bone2d_node: NodePath)
void	set_jiggle_joint_bone_index(joint_idx: int, bone_idx: int)
void	set_jiggle_joint_damping(joint_idx: int, damping: float)
void	set_jiggle_joint_gravity(joint_idx: int, gravity: Vector2)
void	set_jiggle_joint_mass(joint_idx: int, mass: float)

void	set_jiggle_joint_override(joint_idx: int, override: bool)
void	set_jiggle_joint_stiffness(joint_idx: int, stiffness: float)
void	set_jiggle_joint_use_gravity(joint_idx: int, use_gravity: bool)
void	set_use_colliders(use_colliders: bool)

Property Descriptions

float **damping** = `0.75` 🔗

- void **set_damping**(value: float)
- float **get_damping**()

The default amount of damping applied to the Jiggle joints, if they are not overridden. Higher values lead to more of the calculated velocity being applied.

Vector2 **gravity** = `Vector2(0, 6)` 🔗

- void **set_gravity**(value: Vector2)
- Vector2 **get_gravity**()

The default amount of gravity applied to the Jiggle joints, if they are not overridden.

---

int **jiggle_data_chain_length** = `0`

- void **set_jiggle_data_chain_length**(value: int)
- int **get_jiggle_data_chain_length**()

The amount of Jiggle joints in the Jiggle modification.

---

float **mass** = `0.75`

- void **set_mass**(value: float)
- float **get_mass**()

The default amount of mass assigned to the Jiggle joints, if they are not overridden. Higher values lead to faster movements and more overshooting.

---

float **stiffness** = `3.0`

- void **set_stiffness**(value: float)
- float **get_stiffness**()

The default amount of stiffness assigned to the Jiggle joints, if they are not overridden. Higher values act more like springs, quickly moving into the correct position.

NodePath **target_nodepath** = `NodePath("")`

- void **set_target_node**(value: NodePath)
- NodePath **get_target_node**()

The NodePath to the node that is the target for the Jiggle modification. This node is what the Jiggle chain will attempt to rotate the bone chain to.

---

bool **use_gravity** = `false`

- void **set_use_gravity**(value: bool)
- bool **get_use_gravity**()

Whether the gravity vector, gravity, should be applied to the Jiggle joints, assuming they are not overriding the default settings.

Method Descriptions

int **get_collision_mask**() const

Returns the collision mask used by the Jiggle modifier when collisions are enabled.

NodePath **get_jiggle_joint_bone2d_node**(joint_idx: int) const

Returns the Bone2D node assigned to the Jiggle joint at `joint_idx`.

---

int **get_jiggle_joint_bone_index**(joint_idx: int) const

Returns the index of the Bone2D node assigned to the Jiggle joint at `joint_idx`.

---

float **get_jiggle_joint_damping**(joint_idx: int) const

Returns the amount of damping of the Jiggle joint at `joint_idx`.

Vector2 **get_jiggle_joint_gravity**(joint_idx: int) const

Returns a Vector2 representing the amount of gravity the Jiggle joint at `joint_idx` is influenced by.

---

float **get_jiggle_joint_mass**(joint_idx: int) const

Returns the amount of mass of the jiggle joint at `joint_idx`.

---

bool **get_jiggle_joint_override**(joint_idx: int) const

Returns a boolean that indicates whether the joint at `joint_idx` is overriding the default Jiggle joint data defined in the modification.

---

float **get_jiggle_joint_stiffness**(joint_idx: int) const

Returns the stiffness of the Jiggle joint at `joint_idx`.

---

bool **get_jiggle_joint_use_gravity**(joint_idx: int) const

Returns a boolean that indicates whether the joint at `joint_idx` is using gravity or not.

---

bool **get_use_colliders**() const

Returns whether the jiggle modifier is taking physics colliders into account when solving.

---

void **set_collision_mask**(collision_mask: int)

Sets the collision mask that the Jiggle modifier will use when reacting to colliders, if the Jiggle modifier is set to take colliders into account.

---

void **set_jiggle_joint_bone2d_node**(joint_idx: int, bone2d_node: NodePath)

Sets the Bone2D node assigned to the Jiggle joint at `joint_idx`.

---

void **set_jiggle_joint_bone_index**(joint_idx: int, bone_idx: int)

Sets the bone index, `bone_idx`, of the Jiggle joint at `joint_idx`. When possible, this will also update the `bone2d_node` of the Jiggle joint based on data provided by the linked skeleton.

---

void **set_jiggle_joint_damping**(joint_idx: int, damping: float)

Sets the amount of damping of the Jiggle joint at `joint_idx`.

---

void **set_jiggle_joint_gravity**(joint_idx: int, gravity: Vector2)

Sets the gravity vector of the Jiggle joint at `joint_idx`.

---

void **set_jiggle_joint_mass**(joint_idx: int, mass: float)

Sets the of mass of the Jiggle joint at `joint_idx`.

---

void **set_jiggle_joint_override**(joint_idx: int, override: bool)

Sets whether the Jiggle joint at `joint_idx` should override the default Jiggle joint settings. Setting this to `true` will make the joint use its own settings rather than the default ones attached to the modification.

---

void **set_jiggle_joint_stiffness**(joint_idx: int, stiffness: float) 🔗

Sets the of stiffness of the Jiggle joint at `joint_idx`.

---

void **set_jiggle_joint_use_gravity**(joint_idx: int, use_gravity: bool) 🔗

Sets whether the Jiggle joint at `joint_idx` should use gravity.

---

void **set_use_colliders**(use_colliders: bool) 🔗

If `true`, the Jiggle modifier will take colliders into account, keeping them from entering into these collision objects.

SkeletonModification2DLookAt

Experimental: This class may be changed or removed in future versions.

Inherits: SkeletonModification2D **<** Resource **<** RefCounted **<** Object

A modification that rotates a Bone2D node to look at a target.

Description

This SkeletonModification2D rotates a bone to look a target. This is extremely helpful for moving character's head to look at the player, rotating a turret to look at a target, or any other case where you want to make a bone rotate towards something quickly and easily.

Properties

NodePath	bone2d_node	`NodePath("")`
int	bone_index	`-1`
NodePath	target_nodepath	`NodePath("")`

Methods

float	get_additional_rotation() const
bool	get_constraint_angle_invert() const
float	get_constraint_angle_max() const
float	get_constraint_angle_min() const
bool	get_enable_constraint() const
void	set_additional_rotation(rotation: float)
void	set_constraint_angle_invert(invert: bool)
void	set_constraint_angle_max(angle_max: float)
void	set_constraint_angle_min(angle_min: float)
void	set_enable_constraint(enable_constraint: bool)

Property Descriptions

NodePath **bone2d_node** = `NodePath("")`

- void **set_bone2d_node**(value: NodePath)
- NodePath **get_bone2d_node**()

The Bone2D node that the modification will operate on.

---

int **bone_index** = `-1`

- void **set_bone_index**(value: int)
- int **get_bone_index**()

The index of the Bone2D node that the modification will operate on.

NodePath **target_nodepath** = `NodePath("")`

- void **set_target_node**(value: NodePath)
- NodePath **get_target_node**()

The NodePath to the node that is the target for the LookAt modification. This node is what the modification will rotate the Bone2D to.

Method Descriptions

float **get_additional_rotation**() const

Returns the amount of additional rotation that is applied after the LookAt modification executes.

---

bool **get_constraint_angle_invert**() const

Returns whether the constraints to this modification are inverted or not.

---

float **get_constraint_angle_max**() const

Returns the constraint's maximum allowed angle.

---

float **get_constraint_angle_min**() const

Returns the constraint's minimum allowed angle.

---

bool **get_enable_constraint**() const

Returns `true` if the LookAt modification is using constraints.

---

void **set_additional_rotation**(rotation: float)

Sets the amount of additional rotation that is to be applied after executing the modification. This allows for offsetting the results by the inputted rotation amount.

---

void **set_constraint_angle_invert**(invert: bool)

When `true`, the modification will use an inverted joint constraint.

An inverted joint constraint only constraints the Bone2D to the angles *outside of* the inputted minimum and maximum angles. For this reason, it is referred to as an inverted joint constraint, as it constraints the joint to the outside of the inputted values.

---

void **set_constraint_angle_max**(angle_max: float)

Sets the constraint's maximum allowed angle.

void **set_constraint_angle_min**(angle_min: float)

Sets the constraint's minimum allowed angle.

void **set_enable_constraint**(enable_constraint: bool)

Sets whether this modification will use constraints or not. When `true`, constraints will be applied when solving the LookAt modification.

SkeletonModification2DPhysicalBones

Experimental: Physical bones may be changed in the future to perform the position update of Bone2D on their own, without needing this resource.

Inherits: SkeletonModification2D **<** Resource **<** RefCounted **<** Object

A modification that applies the transforms of PhysicalBone2D nodes to Bone2D nodes.

Description

This modification takes the transforms of PhysicalBone2D nodes and applies them to Bone2D nodes. This allows the Bone2D nodes to react to physics thanks to the linked PhysicalBone2D nodes.

Properties

int	physical_bone_chain_length	0

Methods

void	fetch_physical_bones()

NodePath	get_physical_bone_node(joint_idx: int) const
void	set_physical_bone_node(joint_idx: int, physicalbone2d_node: NodePath)
void	start_simulation(bones: Array[StringName] = [])
void	stop_simulation(bones: Array[StringName] = [])

Property Descriptions

int **physical_bone_chain_length** = `0` 🔗

- void **set_physical_bone_chain_length**(value: int)
- int **get_physical_bone_chain_length**()

The number of PhysicalBone2D nodes linked in this modification.

Method Descriptions

void **fetch_physical_bones**() 🔗

Empties the list of PhysicalBone2D nodes and populates it with all PhysicalBone2D nodes that are children of the Skeleton2D.

NodePath **get_physical_bone_node**(joint_idx: int) const 🔗

Returns the PhysicalBone2D node at `joint_idx`.

---

void **set_physical_bone_node**(joint_idx: int, physicalbone2d_node: NodePath)

Sets the PhysicalBone2D node at `joint_idx`.

Note: This is just the index used for this modification, not the bone index used in the Skeleton2D.

---

void **start_simulation**(bones: Array[StringName] = [])

Tell the PhysicalBone2D nodes to start simulating and interacting with the physics world.

Optionally, an array of bone names can be passed to this function, and that will cause only PhysicalBone2D nodes with those names to start simulating.

---

void **stop_simulation**(bones: Array[StringName] = [])

Tell the PhysicalBone2D nodes to stop simulating and interacting with the physics world.

Optionally, an array of bone names can be passed to this function, and that will cause only PhysicalBone2D nodes with those names to stop simulating.

SkeletonModification2DStackHolder

Experimental: This class may be changed or removed in future versions.

Inherits: SkeletonModification2D **<** Resource **<** RefCounted **<** Object

A modification that holds and executes a SkeletonModificationStack2D.

Description

This SkeletonModification2D holds a reference to a SkeletonModificationStack2D, allowing you to use multiple modification stacks on a single Skeleton2D.

Note: The modifications in the held SkeletonModificationStack2D will only be executed if their execution mode matches the execution mode of the SkeletonModification2DStackHolder.

Methods

SkeletonModificationStack2D	get_held_modification_stack() const
void	set_held_modification_stack(held_modification_stack: SkeletonModificationStack2D)

Method Descriptions

SkeletonModificationStack2D **get_held_modification_stack**() const

Returns the SkeletonModificationStack2D that this modification is holding.

---

void **set_held_modification_stack**(held_modification_stack: SkeletonModificationStack2D)

Sets the SkeletonModificationStack2D that this modification is holding. This modification stack will then be executed when this modification is executed.

SkeletonModification2DTwoBoneIK

Experimental: This class may be changed or removed in future versions.

Inherits: SkeletonModification2D **<** Resource **<** RefCounted **<** Object

A modification that rotates two bones using the law of cosines to reach the target.

Description

This SkeletonModification2D uses an algorithm typically called TwoBoneIK. This algorithm works by leveraging the law of cosines and the lengths of the bones to figure out what rotation the bones currently have, and what rotation they need to make a complete triangle, where the first bone, the second bone, and the target form the three vertices of the triangle. Because the algorithm works by making a triangle, it can only operate on two bones.

TwoBoneIK is great for arms, legs, and really any joints that can be represented by just two bones that bend to reach a target. This solver is more lightweight than SkeletonModification2DFABRIK, but gives similar, natural looking results.

Properties

bool	flip_bend_direction	`false`

float	target_maximum_distance	`0.0`
float	target_minimum_distance	`0.0`
NodePath	target_nodepath	`NodePath("")`

Methods

NodePath	get_joint_one_bone2d_node() const
int	get_joint_one_bone_idx() const
NodePath	get_joint_two_bone2d_node() const
int	get_joint_two_bone_idx() const
void	set_joint_one_bone2d_node(bone2d_node: NodePath)
void	set_joint_one_bone_idx(bone_idx: int)
void	set_joint_two_bone2d_node(bone2d_node: NodePath)
void	set_joint_two_bone_idx(bone_idx: int)

Property Descriptions

bool **flip_bend_direction** = `false`

- void **set_flip_bend_direction**(value: bool)
- bool **get_flip_bend_direction**()

If `true`, the bones in the modification will blend outward as opposed to inwards when contracting. If `false`, the bones will bend inwards when contracting.

---

float **target_maximum_distance** = `0.0`

- void **set_target_maximum_distance**(value: float)
- float **get_target_maximum_distance**()

The maximum distance the target can be at. If the target is farther than this distance, the modification will solve as if it's at this maximum distance. When set to `0`, the modification will solve without distance constraints.

---

float **target_minimum_distance** = `0.0`

- void **set_target_minimum_distance**(value: float)
- float **get_target_minimum_distance**()

The minimum distance the target can be at. If the target is closer than this distance, the modification will solve as if it's at this minimum distance. When set to `0`, the modification will solve without distance constraints.

NodePath **target_nodepath** = `NodePath("")`

- void **set_target_node**(value: NodePath)
- NodePath **get_target_node**()

The NodePath to the node that is the target for the TwoBoneIK modification. This node is what the modification will use when bending the Bone2D nodes.

Method Descriptions

NodePath **get_joint_one_bone2d_node**() const

Returns the Bone2D node that is being used as the first bone in the TwoBoneIK modification.

---

int **get_joint_one_bone_idx**() const

Returns the index of the Bone2D node that is being used as the first bone in the TwoBoneIK modification.

NodePath **get_joint_two_bone2d_node**() const

Returns the Bone2D node that is being used as the second bone in the TwoBoneIK modification.

---

int **get_joint_two_bone_idx**() const

Returns the index of the Bone2D node that is being used as the second bone in the TwoBoneIK modification.

---

void **set_joint_one_bone2d_node**(bone2d_node: NodePath)

Sets the Bone2D node that is being used as the first bone in the TwoBoneIK modification.

---

void **set_joint_one_bone_idx**(bone_idx: int)

Sets the index of the Bone2D node that is being used as the first bone in the TwoBoneIK modification.

---

void **set_joint_two_bone2d_node**(bone2d_node: NodePath)

Sets the Bone2D node that is being used as the second bone in the TwoBoneIK modification.

---

void **set_joint_two_bone_idx**(bone_idx: int)

Sets the index of the Bone2D node that is being used as the second bone in the TwoBoneIK modification.

SkeletonModificationStack2D

Experimental: This class may be changed or removed in future versions.

Inherits: Resource **<** RefCounted **<** Object

A resource that holds a stack of SkeletonModification2Ds.

Description

This resource is used by the Skeleton and holds a stack of SkeletonModification2Ds.

This controls the order of the modifications and how they are applied. Modification order is especially important for full-body IK setups, as you need to execute the modifications in the correct order to get the desired results. For example, you want to execute a modification on the spine *before* the arms on a humanoid skeleton.

This resource also controls how strongly all of the modifications are applied to the Skeleton2D.

Properties

bool	enabled	`false`
int	modification_count	`0`

float	strength	`1.0`

Methods

void	add_modification(modification: SkeletonModification2D)
void	delete_modification(mod_idx: int)
void	enable_all_modifications(enabled: bool)
void	execute(delta: float, execution_mode: int)
bool	get_is_setup() const
SkeletonModification2D	get_modification(mod_idx: int) const
Skeleton2D	get_skeleton() const
void	set_modification(mod_idx: int, modification: SkeletonModification2D)
void	setup()

Property Descriptions

bool **enabled** = `false`

- void **set_enabled**(value: bool)
- bool **get_enabled**()

If `true`, the modification's in the stack will be called. This is handled automatically through the Skeleton2D node.

---

int **modification_count** = `0`

- void **set_modification_count**(value: int)
- int **get_modification_count**()

The number of modifications in the stack.

---

float **strength** = `1.0`

- void **set_strength**(value: float)
- float **get_strength**()

The interpolation strength of the modifications in stack. A value of `0` will make it where the modifications are not applied, a strength of `0.5` will be half applied, and a strength of `1` will allow the modifications to be fully applied and override the Skeleton2D Bone2D poses.

Method Descriptions

void **add_modification**(modification: SkeletonModification2D)

Adds the passed-in SkeletonModification2D to the stack.

void **delete_modification**(mod_idx: int)

Deletes the SkeletonModification2D at the index position `mod_idx`, if it exists.

void **enable_all_modifications**(enabled: bool)

Enables all SkeletonModification2Ds in the stack.

void **execute**(delta: float, execution_mode: int)

Executes all of the SkeletonModification2Ds in the stack that use the same execution mode as the passed-in `execution_mode`, starting from index `0` to modification_count.

Note: The order of the modifications can matter depending on the modifications. For example, modifications on a spine should operate before modifications on the arms in order to get proper results.

bool **get_is_setup**() const

Returns a boolean that indicates whether the modification stack is setup and can execute.

SkeletonModification2D **get_modification**(mod_idx: int) const

Returns the SkeletonModification2D at the passed-in index, `mod_idx`.

Skeleton2D **get_skeleton**() const

Returns the Skeleton2D node that the SkeletonModificationStack2D is bound to.

void **set_modification**(mod_idx: int, modification: SkeletonModification2D)

Sets the modification at `mod_idx` to the passed-in modification, `modification`.

void **setup**()

Sets up the modification stack so it can execute. This function should be called by Skeleton2D and shouldn't be manually called unless you know what you are doing.

SkeletonProfile

Inherits: Resource **<** RefCounted **<** Object

Inherited By: SkeletonProfileHumanoid

Base class for a profile of a virtual skeleton used as a target for retargeting.

Description

This resource is used in EditorScenePostImport. Some parameters are referring to bones in Skeleton3D, Skin, Animation, and some other nodes are rewritten based on the parameters of **SkeletonProfile**.

Note: These parameters need to be set only when creating a custom profile. In SkeletonProfileHumanoid, they are defined internally as read-only values.

Tutorials

- Retargeting 3D Skeletons

Properties

int	bone_size	0
int	group_size	0

StringName	root_bone	`&""`
StringName	scale_base_bone	`&""`

Methods

int	find_bone(bone_name: StringName) const
StringName	get_bone_name(bone_idx: int) const
StringName	get_bone_parent(bone_idx: int) const
StringName	get_bone_tail(bone_idx: int) const
StringName	get_group(bone_idx: int) const
StringName	get_group_name(group_idx: int) const
Vector2	get_handle_offset(bone_idx: int) const
Transform3D	get_reference_pose(bone_idx: int) const
TailDirection	get_tail_direction(bone_idx: int) const

Texture2D	get_texture(group_idx: int) const
bool	is_required(bone_idx: int) const
void	set_bone_name(bone_idx: int, bone_name: StringName)
void	set_bone_parent(bone_idx: int, bone_parent: StringName)
void	set_bone_tail(bone_idx: int, bone_tail: StringName)
void	set_group(bone_idx: int, group: StringName)
void	set_group_name(group_idx: int, group_name: StringName)
void	set_handle_offset(bone_idx: int, handle_offset: Vector2)
void	set_reference_pose(bone_idx: int, bone_name: Transform3D)
void	set_required(bone_idx: int, required: bool)
void	set_tail_direction(bone_idx: int, tail_direction: TailDirection)

void	set_texture(group_idx: int, texture: Texture2D)

Signals

profile_updated()

This signal is emitted when change the value in profile. This is used to update key name in the BoneMap and to redraw the BoneMap editor.

Note: This signal is not connected directly to editor to simplify the reference, instead it is passed on to editor through the BoneMap.

Enumerations

enum **TailDirection**:

TailDirection **TAIL_DIRECTION_AVERAGE_CHILDREN** = `0`

Direction to the average coordinates of bone children.

TailDirection **TAIL_DIRECTION_SPECIFIC_CHILD** = `1`

Direction to the coordinates of specified bone child.

TailDirection **TAIL_DIRECTION_END** = `2`

Direction is not calculated.

Property Descriptions

int **bone_size** = `0` 🔗

- void **set_bone_size**(value: int)
- int **get_bone_size**()

The amount of bones in retargeting section's BoneMap editor. For example, SkeletonProfileHumanoid has 56 bones.

The size of elements in BoneMap updates when changing this property in it's assigned **SkeletonProfile**.

---

int **group_size** = `0` 🔗

- void **set_group_size**(value: int)
- int **get_group_size**()

The amount of groups of bones in retargeting section's BoneMap editor. For example, SkeletonProfileHumanoid has 4 groups.

This property exists to separate the bone list into several sections in the editor.

StringName **root_bone** = `&""` 🔗

- void **set_root_bone**(value: StringName)
- StringName **get_root_bone**()

A bone name that will be used as the root bone in AnimationTree. This should be the bone of the parent of hips that exists at the world origin.

StringName **scale_base_bone** = `&""`

- void **set_scale_base_bone**(value: StringName)
- StringName **get_scale_base_bone**()

A bone name which will use model's height as the coefficient for normalization. For example, SkeletonProfileHumanoid defines it as `Hips`.

Method Descriptions

int **find_bone**(bone_name: StringName) const

Returns the bone index that matches `bone_name` as its name.

StringName **get_bone_name**(bone_idx: int) const

Returns the name of the bone at `bone_idx` that will be the key name in the BoneMap.

In the retargeting process, the returned bone name is the bone name of the target skeleton.

StringName **get_bone_parent**(bone_idx: int) const

Returns the name of the bone which is the parent to the bone at `bone_idx`. The result is empty if the bone has no parent.

StringName **get_bone_tail**(bone_idx: int) const

Returns the name of the bone which is the tail of the bone at `bone_idx`.

StringName **get_group**(bone_idx: int) const

Returns the group of the bone at `bone_idx`.

StringName **get_group_name**(group_idx: int) const

Returns the name of the group at `group_idx` that will be the drawing group in the BoneMap editor.

Vector2 **get_handle_offset**(bone_idx: int) const

Returns the offset of the bone at `bone_idx` that will be the button position in the BoneMap editor.

This is the offset with origin at the top left corner of the square.

Transform3D **get_reference_pose**(bone_idx: int) const

Returns the reference pose transform for bone `bone_idx`.

TailDirection **get_tail_direction**(bone_idx: int) const

Returns the tail direction of the bone at `bone_idx`.

Texture2D **get_texture**(group_idx: int) const

Returns the texture of the group at `group_idx` that will be the drawing group background image in the BoneMap editor.

---

bool **is_required**(bone_idx: int) const

Returns whether the bone at `bone_idx` is required for retargeting.

This value is used by the bone map editor. If this method returns `true`, and no bone is assigned, the handle color will be red on the bone map editor.

---

void **set_bone_name**(bone_idx: int, bone_name: StringName)

Sets the name of the bone at `bone_idx` that will be the key name in the BoneMap.

In the retargeting process, the setting bone name is the bone name of the target skeleton.

---

void **set_bone_parent**(bone_idx: int, bone_parent: StringName)

Sets the bone with name `bone_parent` as the parent of the bone at `bone_idx`. If an empty string is passed, then the bone has no parent.

---

void **set_bone_tail**(bone_idx: int, bone_tail: StringName)

Sets the bone with name `bone_tail` as the tail of the bone at `bone_idx`.

---

void **set_group**(bone_idx: int, group: StringName)

Sets the group of the bone at `bone_idx`.

---

void **set_group_name**(group_idx: int, group_name: StringName)

Sets the name of the group at `group_idx` that will be the drawing group in the BoneMap editor.

void **set_handle_offset**(bone_idx: int, handle_offset: Vector2)

Sets the offset of the bone at `bone_idx` that will be the button position in the BoneMap editor.

This is the offset with origin at the top left corner of the square.

void **set_reference_pose**(bone_idx: int, bone_name: Transform3D)

Sets the reference pose transform for bone `bone_idx`.

void **set_required**(bone_idx: int, required: bool)

Sets the required status for bone `bone_idx` to `required`.

void **set_tail_direction**(bone_idx: int, tail_direction: TailDirection)

Sets the tail direction of the bone at `bone_idx`.

Note: This only specifies the method of calculation. The actual coordinates required should be stored in an external skeleton, so the calculation itself needs to be done externally.

void **set_texture**(group_idx: int, texture: Texture2D)

Sets the texture of the group at `group_idx` that will be the drawing group background image in the BoneMap editor.

SkeletonProfileHumanoid

Inherits: SkeletonProfile **<** Resource **<** RefCounted **<** Object

A humanoid SkeletonProfile preset.

Description

A SkeletonProfile as a preset that is optimized for the human form. This exists for standardization, so all parameters are read-only.

A humanoid skeleton profile contains 54 bones divided in 4 groups: `"Body"`, `"Face"`, `"LeftHand"`, and `"RightHand"`. It is structured as follows:

```
Root
└─ Hips
    ├─ LeftUpperLeg
    │   └─ LeftLowerLeg
    │       └─ LeftFoot
    │           └─ LeftToes
    ├─ RightUpperLeg
    │   └─ RightLowerLeg
    │       └─ RightFoot
    │           └─ RightToes
    └─ Spine
        └─ Chest
            └─ UpperChest
                ├─ Neck
                │   └─ Head
                │       ├─ Jaw
                │       ├─ LeftEye
                │       └─ RightEye
                ├─ LeftShoulder
                │  └─ LeftUpperArm
                │     └─ LeftLowerArm
                │
```

```
│        └─ LeftHand
│           ├─ LeftThumbMetacarpal
│           │  └─ LeftThumbProximal
│           ├─ LeftIndexProximal
│           │  └─ LeftIndexIntermediate
│           │     └─ LeftIndexDistal
│           ├─ LeftMiddleProximal
│           │  └─ LeftMiddleIntermediate
│           │     └─ LeftMiddleDistal
│           ├─ LeftRingProximal
│           │  └─ LeftRingIntermediate
│           │     └─ LeftRingDistal
│           └─ LeftLittleProximal
│              └─ LeftLittleIntermediate
│                 └─ LeftLittleDistal
└─ RightShoulder
   └─ RightUpperArm
      └─ RightLowerArm
         └─ RightHand
            ├─ RightThumbMetacarpal
            │  └─ RightThumbProximal
            ├─ RightIndexProximal
            │  └─ RightIndexIntermediate
            │      └─ RightIndexDistal
            ├─ RightMiddleProximal
            │  └─ RightMiddleIntermediate
            │      └─ RightMiddleDistal
            ├─ RightRingProximal
            │  └─ RightRingIntermediate
            │      └─ RightRingDistal
            └─ RightLittleProximal
               └─ RightLittleIntermediate
                  └─ RightLittleDistal
```

Tutorials

- Retargeting 3D Skeletons

Properties

int	bone_size	`56` (overrides SkeletonProfile)
int	group_size	`4` (overrides SkeletonProfile)
StringName	root_bone	`&"Root"` (overrides SkeletonProfile)
StringName	scale_base_bone	`&"Hips"` (overrides SkeletonProfile)

Skin

Inherits: Resource **<** RefCounted **<** Object

There is currently no description for this class. Please help us by contributing one!

Methods

void	add_bind(bone: int, pose: Transform3D)
void	add_named_bind(name: String, pose: Transform3D)
void	clear_binds()
int	get_bind_bone(bind_index: int) const
int	get_bind_count() const
StringName	get_bind_name(bind_index: int) const
Transform3D	get_bind_pose(bind_index: int) const
void	set_bind_bone(bind_index: int, bone: int)

void	set_bind_count(bind_count: int)
void	set_bind_name(bind_index: int, name: StringName)
void	set_bind_pose(bind_index: int, pose: Transform3D)

Method Descriptions

void **add_bind**(bone: int, pose: Transform3D) 🔗

There is currently no description for this method. Please help us by contributing one!

---

void **add_named_bind**(name: String, pose: Transform3D) 🔗

There is currently no description for this method. Please help us by contributing one!

---

void **clear_binds**() 🔗

There is currently no description for this method. Please help us by contributing one!

---

int **get_bind_bone**(bind_index: int) const 🔗

There is currently no description for this method. Please help us by contributing one!

int **get_bind_count**() const

There is currently no description for this method. Please help us by contributing one!

StringName **get_bind_name**(bind_index: int) const

There is currently no description for this method. Please help us by contributing one!

Transform3D **get_bind_pose**(bind_index: int) const

There is currently no description for this method. Please help us by contributing one!

---

void **set_bind_bone**(bind_index: int, bone: int)

There is currently no description for this method. Please help us by contributing one!

---

void **set_bind_count**(bind_count: int)

There is currently no description for this method. Please help us by contributing one!

---

void **set_bind_name**(bind_index: int, name: StringName)

There is currently no description for this method. Please help us by contributing one!

---

void **set_bind_pose**(bind_index: int, pose: Transform3D)

There is currently no description for this method. Please help us by contributing one!

Sky

Inherits: Resource **<** RefCounted **<** Object

Defines a 3D environment's background by using a Material.

Description

The **Sky** class uses a Material to render a 3D environment's background and the light it emits by updating the reflection/radiance cubemaps.

Properties

ProcessMode	process_mode	`0`
RadianceSize	radiance_size	`3`
Material	sky_material	

Enumerations

enum **RadianceSize**: 🔗

RadianceSize **RADIANCE_SIZE_32** = `0`

Radiance texture size is 32×32 pixels.

RadianceSize **RADIANCE_SIZE_64** = `1`

Radiance texture size is 64×64 pixels.

RadianceSize **RADIANCE_SIZE_128** = `2`

Radiance texture size is 128×128 pixels.

RadianceSize **RADIANCE_SIZE_256** = `3`

Radiance texture size is 256×256 pixels.

RadianceSize **RADIANCE_SIZE_512** = `4`

Radiance texture size is 512×512 pixels.

RadianceSize **RADIANCE_SIZE_1024** = `5`

Radiance texture size is 1024×1024 pixels.

RadianceSize **RADIANCE_SIZE_2048** = `6`

Radiance texture size is 2048×2048 pixels.

RadianceSize **RADIANCE_SIZE_MAX** = `7`

Represents the size of the RadianceSize enum.

---

enum **ProcessMode**: 🔗

ProcessMode **PROCESS_MODE_AUTOMATIC** = `0`

Automatically selects the appropriate process mode based on your sky shader. If your shader uses `TIME` or `POSITION`, this will use PROCESS_MODE_REALTIME. If your shader uses any of the `LIGHT_*` variables or any custom uniforms, this uses

PROCESS_MODE_INCREMENTAL. Otherwise, this defaults to PROCESS_MODE_QUALITY.

ProcessMode **PROCESS_MODE_QUALITY** = `1`

Uses high quality importance sampling to process the radiance map. In general, this results in much higher quality than PROCESS_MODE_REALTIME but takes much longer to generate. This should not be used if you plan on changing the sky at runtime. If you are finding that the reflection is not blurry enough and is showing sparkles or fireflies, try increasing ProjectSettings.rendering/reflections/sky_reflections/ggx_samples.

ProcessMode **PROCESS_MODE_INCREMENTAL** = `2`

Uses the same high quality importance sampling to process the radiance map as PROCESS_MODE_QUALITY, but updates over several frames. The number of frames is determined by ProjectSettings.rendering/reflections/sky_reflections/roughness_layers. Use this when you need highest quality radiance maps, but have a sky that updates slowly.

ProcessMode **PROCESS_MODE_REALTIME** = `3`

Uses the fast filtering algorithm to process the radiance map. In general this results in lower quality, but substantially faster run times. If you need better quality, but still need to update the sky every frame, consider turning on ProjectSettings.rendering/reflections/sky_reflections/fast_filter_high_quality.

Note: The fast filtering algorithm is limited to 256×256 cubemaps, so radiance_size must be set to RADIANCE_SIZE_256. Otherwise, a warning is printed and the overridden radiance size is ignored.

Property Descriptions

ProcessMode **process_mode** = 0 🔗

- void **set_process_mode**(value: ProcessMode)
- ProcessMode **get_process_mode**()

Sets the method for generating the radiance map from the sky. The radiance map is a cubemap with increasingly blurry versions of the sky corresponding to different levels of roughness. Radiance maps can be expensive to calculate. See ProcessMode for options.

RadianceSize **radiance_size** = 3 🔗

- void **set_radiance_size**(value: RadianceSize)
- RadianceSize **get_radiance_size**()

The **Sky**'s radiance map size. The higher the radiance map size, the more detailed the lighting from the **Sky** will be.

See RadianceSize constants for values.

Note: Some hardware will have trouble with higher radiance sizes, especially RADIANCE_SIZE_512 and above. Only use such high values on high-end hardware.

Material **sky_material** 🔗

- void **set_material**(value: Material)
- Material **get_material**()

Material used to draw the background. Can be PanoramaSkyMaterial, ProceduralSkyMaterial, PhysicalSkyMaterial, or even a ShaderMaterial if you want to use your own custom shader.

SphereMesh

Inherits: PrimitiveMesh **<** Mesh **<** Resource **<** RefCounted **<** Object

Class representing a spherical PrimitiveMesh.

Description

Class representing a spherical PrimitiveMesh.

Properties

float	height	`1.0`
bool	is_hemisphere	`false`
int	radial_segments	`64`
float	radius	`0.5`
int	rings	`32`

Property Descriptions

float **height** = `1.0` 🔗

- void **set_height**(value: float)
- float **get_height**()

Full height of the sphere.

---

bool **is_hemisphere** = `false` 🔗

- void **set_is_hemisphere**(value: bool)
- bool **get_is_hemisphere**()

If `true`, a hemisphere is created rather than a full sphere.

Note: To get a regular hemisphere, the height and radius of the sphere must be equal.

---

int **radial_segments** = `64` 🔗

- void **set_radial_segments**(value: int)
- int **get_radial_segments**()

Number of radial segments on the sphere.

---

float **radius** = `0.5` 🔗

- void **set_radius**(value: float)
- float **get_radius**()

Radius of sphere.

---

int **rings** = `32` 🔗

- void **set_rings**(value: int)
- int **get_rings**()

Number of segments along the height of the sphere.

SphereOccluder3D

Inherits: Occluder3D **<** Resource **<** RefCounted **<** Object

Spherical shape for use with occlusion culling in OccluderInstance3D.

Description

SphereOccluder3D stores a sphere shape that can be used by the engine's occlusion culling system.

See OccluderInstance3D's documentation for instructions on setting up occlusion culling.

Tutorials

- Occlusion culling

Properties

float	radius	`1.0`

Property Descriptions

float **radius** = `1.0` 🔗

- void **set_radius**(value: float)

- float **get_radius**()

The sphere's radius in 3D units.

SphereShape3D

Inherits: Shape3D **<** Resource **<** RefCounted **<** Object

A 3D sphere shape used for physics collision.

Description

A 3D sphere shape, intended for use in physics. Usually used to provide a shape for a CollisionShape3D.

Performance: SphereShape3D is fast to check collisions against. It is faster than BoxShape3D, CapsuleShape3D, and CylinderShape3D.

Tutorials

- 3D Physics Tests Demo [https://godotengine.org/asset-library/asset/2747]

Properties

float	radius	`0.5`

Property Descriptions

float **radius** = `0.5` 🔗

- void **set_radius**(value: float)
- float **get_radius**()

The sphere's radius. The shape's diameter is double the radius.

SpriteFrames

Inherits: Resource **<** RefCounted **<** Object

Sprite frame library for AnimatedSprite2D and AnimatedSprite3D.

Description

Sprite frame library for an AnimatedSprite2D or AnimatedSprite3D node. Contains frames and animation data for playback.

Methods

void	add_animation(anim: StringName)
void	add_frame(anim: StringName, texture: Texture2D, duration: float = 1.0, at_position: int = -1)
void	clear(anim: StringName)
void	clear_all()
bool	get_animation_loop(anim: StringName) const
PackedStringArray	get_animation_names() const

float	get_animation_speed(anim: StringName) const
int	get_frame_count(anim: StringName) const
float	get_frame_duration(anim: StringName, idx: int) const
Texture2D	get_frame_texture(anim: StringName, idx: int) const
bool	has_animation(anim: StringName) const
void	remove_animation(anim: StringName)
void	remove_frame(anim: StringName, idx: int)
void	rename_animation(anim: StringName, newname: StringName)
void	set_animation_loop(anim: StringName, loop: bool)
void	set_animation_speed(anim: StringName, fps: float)
void	set_frame(anim: StringName, idx: int, texture: Texture2D, duration: float = 1.0)

Method Descriptions

void **add_animation**(anim: StringName) 🔗

Adds a new `anim` animation to the library.

void **add_frame**(anim: StringName, texture: Texture2D, duration: float = 1.0, at_position: int = -1) 🔗

Adds a frame to the `anim` animation. If `at_position` is `-1`, the frame will be added to the end of the animation. `duration` specifies the relative duration, see get_frame_duration for details.

void **clear**(anim: StringName) 🔗

Removes all frames from the `anim` animation.

void **clear_all**() 🔗

Removes all animations. An empty `default` animation will be created.

bool **get_animation_loop**(anim: StringName) const 🔗

Returns `true` if the given animation is configured to loop when it finishes playing. Otherwise, returns `false`.

PackedStringArray **get_animation_names**() const 🔗

Returns an array containing the names associated to each animation. Values are placed in alphabetical order.

---

float **get_animation_speed**(anim: StringName) const

Returns the speed in frames per second for the `anim` animation.

---

int **get_frame_count**(anim: StringName) const

Returns the number of frames for the `anim` animation.

---

float **get_frame_duration**(anim: StringName, idx: int) const

Returns a relative duration of the frame `idx` in the `anim` animation (defaults to `1.0`). For example, a frame with a duration of `2.0` is displayed twice as long as a frame with a duration of `1.0`. You can calculate the absolute duration (in seconds) of a frame using the following formula:

```
absolute_duration = relative_duration / (animation_fps *
abs(playing_speed))
```

In this example, `playing_speed` refers to either AnimatedSprite2D.get_playing_speed or AnimatedSprite3D.get_playing_speed.

Texture2D **get_frame_texture**(anim: StringName, idx: int) const

Returns the texture of the frame `idx` in the `anim` animation.

---

bool **has_animation**(anim: StringName) const

Returns `true` if the `anim` animation exists.

---

void **remove_animation**(anim: StringName)

Removes the `anim` animation.

---

void **remove_frame**(anim: StringName, idx: int)

Removes the `anim` animation's frame `idx`.

---

void **rename_animation**(anim: StringName, newname: StringName)

Changes the `anim` animation's name to `newname`.

---

void **set_animation_loop**(anim: StringName, loop: bool)

If `loop` is `true`, the `anim` animation will loop when it reaches the end, or the start if it is played in reverse.

---

void **set_animation_speed**(anim: StringName, fps: float)

Sets the speed for the `anim` animation in frames per second.

---

void **set_frame**(anim: StringName, idx: int, texture: Texture2D, duration: float = 1.0)

Sets the `texture` and the `duration` of the frame `idx` in the `anim` animation. `duration` specifies the relative duration, see get_frame_duration for details.

StandardMaterial3D

Inherits: BaseMaterial3D **<** Material **<** Resource **<** RefCounted **<** Object

A PBR (Physically Based Rendering) material to be used on 3D objects.

Description

StandardMaterial3D's properties are inherited from BaseMaterial3D. **StandardMaterial3D** uses separate textures for ambient occlusion, roughness and metallic maps. To use a single ORM map for all 3 textures, use an ORMMaterial3D instead.

Tutorials

- Standard Material 3D and ORM Material 3D

StyleBox

Inherits: Resource **<** RefCounted **<** Object

Inherited By: StyleBoxEmpty, StyleBoxFlat, StyleBoxLine, StyleBoxTexture

Abstract base class for defining stylized boxes for UI elements.

Description

StyleBox is an abstract base class for drawing stylized boxes for UI elements. It is used for panels, buttons, LineEdit backgrounds, Tree backgrounds, etc. and also for testing a transparency mask for pointer signals. If mask test fails on a **StyleBox** assigned as mask to a control, clicks and motion signals will go through it to the one below.

Note: For control nodes that have *Theme Properties*, the `focus` **StyleBox** is displayed over the `normal`, `hover` or `pressed` **StyleBox**. This makes the `focus` **StyleBox** more reusable across different nodes.

Properties

float	content_margin_bottom	`-1.0`
float	content_margin_left	`-1.0`

float	content_margin_right	`-1.0`
float	content_margin_top	`-1.0`

Methods

void	_draw(to_canvas_item: RID, rect: Rect2) virtual const
Rect2	_get_draw_rect(rect: Rect2) virtual const
Vector2	_get_minimum_size() virtual const
bool	_test_mask(point: Vector2, rect: Rect2) virtual const
void	draw(canvas_item: RID, rect: Rect2) const
float	get_content_margin(margin: Side) const
CanvasItem	get_current_item_drawn() const
float	get_margin(margin: Side) const
Vector2	get_minimum_size() const

Vector2	get_offset() const
void	set_content_margin(margin: Side, offset: float)
void	set_content_margin_all(offset: float)
bool	test_mask(point: Vector2, rect: Rect2) const

Property Descriptions

float **content_margin_bottom** = `-1.0`

- void **set_content_margin**(margin: Side, offset: float)
- float **get_content_margin**(margin: Side) const

The bottom margin for the contents of this style box. Increasing this value reduces the space available to the contents from the bottom.

If this value is negative, it is ignored and a child-specific margin is used instead. For example, for StyleBoxFlat, the border thickness (if any) is used instead.

It is up to the code using this style box to decide what these contents are: for example, a Button respects this content margin for the textual contents of the button.

get_margin should be used to fetch this value as consumer instead of reading these properties directly. This is because it correctly respects negative values and the fallback mentioned above.

float **content_margin_left** = `-1.0` 🔗

- void **set_content_margin**(margin: Side, offset: float)
- float **get_content_margin**(margin: Side) const

The left margin for the contents of this style box. Increasing this value reduces the space available to the contents from the left.

Refer to content_margin_bottom for extra considerations.

---

float **content_margin_right** = `-1.0` 🔗

- void **set_content_margin**(margin: Side, offset: float)
- float **get_content_margin**(margin: Side) const

The right margin for the contents of this style box. Increasing this value reduces the space available to the contents from the right.

Refer to content_margin_bottom for extra considerations.

---

float **content_margin_top** = `-1.0` 🔗

- void **set_content_margin**(margin: Side, offset: float)
- float **get_content_margin**(margin: Side) const

The top margin for the contents of this style box. Increasing this value reduces the space available to the contents from the top.

Refer to content_margin_bottom for extra considerations.

Method Descriptions

void **_draw**(to_canvas_item: RID, rect: Rect2) virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

Rect2 **_get_draw_rect**(rect: Rect2) virtual const

There is currently no description for this method. Please help us by contributing one!

Vector2 **_get_minimum_size**() virtual const

Virtual method to be implemented by the user. Returns a custom minimum size that the stylebox must respect when drawing. By default get_minimum_size only takes content margins into account. This method can be overridden to add another size restriction. A combination of the default behavior and the output of this method will be used, to account for both sizes.

---

bool **_test_mask**(point: Vector2, rect: Rect2) virtual const

There is currently no description for this method. Please help us by contributing one!

---

void **draw**(canvas_item: RID, rect: Rect2) const

Draws this stylebox using a canvas item identified by the given RID.

The RID value can either be the result of CanvasItem.get_canvas_item called on an existing CanvasItem-derived node, or directly from creating a canvas item in the RenderingServer with RenderingServer.canvas_item_create.

---

float **get_content_margin**(margin: Side) const

Returns the default margin of the specified Side.

CanvasItem **get_current_item_drawn**() const

Returns the CanvasItem that handles its CanvasItem.NOTIFICATION_DRAW or CanvasItem._draw callback at this moment.

float **get_margin**(margin: Side) const

Returns the content margin offset for the specified Side.

Positive values reduce size inwards, unlike Control's margin values.

Vector2 **get_minimum_size**() const

Returns the minimum size that this stylebox can be shrunk to.

Vector2 **get_offset**() const

Returns the "offset" of a stylebox. This helper function returns a value equivalent to `Vector2(style.get_margin(MARGIN_LEFT), style.get_margin(MARGIN_TOP))`.

void **set_content_margin**(margin: Side, offset: float)

Sets the default value of the specified Side to `offset` pixels.

void **set_content_margin_all**(offset: float)

Sets the default margin to `offset` pixels for all sides.

---

bool **test_mask**(point: Vector2, rect: Rect2) const

Test a position in a rectangle, return whether it passes the mask test.

StyleBoxEmpty

Inherits: StyleBox **<** Resource **<** RefCounted **<** Object

An empty StyleBox (does not display anything).

Description

An empty StyleBox that can be used to display nothing instead of the default style (e.g. it can "disable" `focus` styles).

StyleBoxFlat

Inherits: StyleBox **<** Resource **<** RefCounted **<** Object

A customizable StyleBox that doesn't use a texture.

Description

By configuring various properties of this style box, you can achieve many common looks without the need of a texture. This includes optionally rounded borders, antialiasing, shadows, and skew.

Setting corner radius to high values is allowed. As soon as corners overlap, the stylebox will switch to a relative system.

Example:

```
height = 30
corner_radius_top_left = 50
corner_radius_bottom_left = 100
```

The relative system now would take the 1:2 ratio of the two left corners to calculate the actual corner width. Both corners added will **never** be more than the height. Result:

```
corner_radius_top_left: 10
corner_radius_bottom_left: 20
```

Properties

bool	anti_aliasing	`true`

float	anti_aliasing_size	`1.0`
Color	bg_color	`Color(0.6, 0.6, 0.6, 1)`
bool	border_blend	`false`
Color	border_color	`Color(0.8, 0.8, 0.8, 1)`
int	border_width_bottom	`0`
int	border_width_left	`0`
int	border_width_right	`0`
int	border_width_top	`0`
int	corner_detail	`8`
int	corner_radius_bottom_left	`0`
int	corner_radius_bottom_right	`0`
int	corner_radius_top_left	`0`
int	corner_radius_top_right	`0`

bool	draw_center	`true`
float	expand_margin_bottom	`0.0`
float	expand_margin_left	`0.0`
float	expand_margin_right	`0.0`
float	expand_margin_top	`0.0`
Color	shadow_color	`Color(0, 0, 0, 0.6)`
Vector2	shadow_offset	`Vector2(0, 0)`
int	shadow_size	`0`
Vector2	skew	`Vector2(0, 0)`

Methods

int	get_border_width(margin: Side) const
int	get_border_width_min() const

int	get_corner_radius(corner: Corner) const
float	get_expand_margin(margin: Side) const
void	set_border_width(margin: Side, width: int)
void	set_border_width_all(width: int)
void	set_corner_radius(corner: Corner, radius: int)
void	set_corner_radius_all(radius: int)
void	set_expand_margin(margin: Side, size: float)
void	set_expand_margin_all(size: float)

Property Descriptions

bool **anti_aliasing** = `true` 🔗

- void **set_anti_aliased**(value: bool)
- bool **is_anti_aliased**()

Antialiasing draws a small ring around the edges, which fades to transparency. As a result, edges look much smoother. This is only noticeable when using rounded corners or skew.

Note: When using beveled corners with 45-degree angles (corner_detail = 1), it is recommended to set anti_aliasing to `false` to ensure crisp visuals and avoid possible visual glitches.

---

float **anti_aliasing_size** = `1.0`

- void **set_aa_size**(value: float)
- float **get_aa_size**()

This changes the size of the antialiasing effect. `1.0` is recommended for an optimal result at 100% scale, identical to how rounded rectangles are rendered in web browsers and most vector drawing software.

Note: Higher values may produce a blur effect but can also create undesired artifacts on small boxes with large-radius corners.

Color **bg_color** = `Color(0.6, 0.6, 0.6, 1)`

- void **set_bg_color**(value: Color)
- Color **get_bg_color**()

The background color of the stylebox.

---

bool **border_blend** = `false`

- void **set_border_blend**(value: bool)
- bool **get_border_blend**()

If `true`, the border will fade into the background color.

Color **border_color** = `Color(0.8, 0.8, 0.8, 1)`

- void **set_border_color**(value: Color)
- Color **get_border_color**()

Sets the color of the border.

int **border_width_bottom** = 0

- void **set_border_width**(margin: Side, width: int)
- int **get_border_width**(margin: Side) const

Border width for the bottom border.

int **border_width_left** = 0

- void **set_border_width**(margin: Side, width: int)
- int **get_border_width**(margin: Side) const

Border width for the left border.

int **border_width_right** = 0

- void **set_border_width**(margin: Side, width: int)
- int **get_border_width**(margin: Side) const

Border width for the right border.

int **border_width_top** = 0

- void **set_border_width**(margin: Side, width: int)
- int **get_border_width**(margin: Side) const

Border width for the top border.

int **corner_detail** = `8` 🔗

- void **set_corner_detail**(value: int)
- int **get_corner_detail**()

This sets the number of vertices used for each corner. Higher values result in rounder corners but take more processing power to compute. When choosing a value, you should take the corner radius (set_corner_radius_all) into account.

For corner radii less than 10, `4` or `5` should be enough. For corner radii less than 30, values between `8` and `12` should be enough.

A corner detail of `1` will result in chamfered corners instead of rounded corners, which is useful for some artistic effects.

---

int **corner_radius_bottom_left** = `0` 🔗

- void **set_corner_radius**(corner: Corner, radius: int)
- int **get_corner_radius**(corner: Corner) const

The bottom-left corner's radius. If `0`, the corner is not rounded.

---

int **corner_radius_bottom_right** = `0` 🔗

- void **set_corner_radius**(corner: Corner, radius: int)
- int **get_corner_radius**(corner: Corner) const

The bottom-right corner's radius. If `0`, the corner is not rounded.

---

int **corner_radius_top_left** = `0` 🔗

- void **set_corner_radius**(corner: Corner, radius: int)
- int **get_corner_radius**(corner: Corner) const

The top-left corner's radius. If `0`, the corner is not rounded.

---

int **corner_radius_top_right** = `0` 🔗

- void **set_corner_radius**(corner: Corner, radius: int)
- int **get_corner_radius**(corner: Corner) const

The top-right corner's radius. If `0`, the corner is not rounded.

---

bool **draw_center** = `true` 🔗

- void **set_draw_center**(value: bool)
- bool **is_draw_center_enabled**()

Toggles drawing of the inner part of the stylebox.

---

float **expand_margin_bottom** = `0.0` 🔗

- void **set_expand_margin**(margin: Side, size: float)
- float **get_expand_margin**(margin: Side) const

Expands the stylebox outside of the control rect on the bottom edge. Useful in combination with border_width_bottom to draw a border outside the control rect.

Note: Unlike StyleBox.content_margin_bottom, expand_margin_bottom does *not* affect the size of the clickable area for Controls. This can negatively impact usability if used wrong, as the user may try to click an area of the StyleBox that cannot actually receive clicks.

---

float **expand_margin_left** = `0.0` 🔗

- void **set_expand_margin**(margin: Side, size: float)
- float **get_expand_margin**(margin: Side) const

Expands the stylebox outside of the control rect on the left edge. Useful in combination with border_width_left to draw a border outside the control rect.

Note: Unlike StyleBox.content_margin_left, expand_margin_left does *not* affect the size of the clickable area for Controls. This can negatively impact usability if used wrong, as the user may try to click an area of the StyleBox that cannot actually receive clicks.

---

float **expand_margin_right** = `0.0`

- void **set_expand_margin**(margin: Side, size: float)
- float **get_expand_margin**(margin: Side) const

Expands the stylebox outside of the control rect on the right edge. Useful in combination with border_width_right to draw a border outside the control rect.

Note: Unlike StyleBox.content_margin_right, expand_margin_right does *not* affect the size of the clickable area for Controls. This can negatively impact usability if used wrong, as the user may try to click an area of the StyleBox that cannot actually receive clicks.

---

float **expand_margin_top** = `0.0`

- void **set_expand_margin**(margin: Side, size: float)
- float **get_expand_margin**(margin: Side) const

Expands the stylebox outside of the control rect on the top edge. Useful in combination with border_width_top to draw a border outside the control rect.

Note: Unlike StyleBox.content_margin_top, expand_margin_top does *not* affect the size of the clickable area for Controls. This can negatively impact usability if used wrong, as the user may try to click an area of the StyleBox that cannot actually receive clicks.

Color **shadow_color** = `Color(0, 0, 0, 0.6)`

- void **set_shadow_color**(value: Color)
- Color **get_shadow_color**()

The color of the shadow. This has no effect if shadow_size is lower than 1.

Vector2 **shadow_offset** = `Vector2(0, 0)`

- void **set_shadow_offset**(value: Vector2)
- Vector2 **get_shadow_offset**()

The shadow offset in pixels. Adjusts the position of the shadow relatively to the stylebox.

---

int **shadow_size** = `0`

- void **set_shadow_size**(value: int)
- int **get_shadow_size**()

The shadow size in pixels.

Vector2 **skew** = `Vector2(0, 0)`

- void **set_skew**(value: Vector2)
- Vector2 **get_skew**()

If set to a non-zero value on either axis, skew distorts the StyleBox horizontally and/or vertically. This can be used for "futuristic"-style UIs. Positive values skew the StyleBox towards the right (X axis) and upwards (Y axis), while negative values skew the StyleBox towards the left (X axis) and downwards (Y axis).

Note: To ensure text does not touch the StyleBox's edges, consider increasing the StyleBox's content margin (see StyleBox.content_margin_bottom). It is preferable to increase the content margin instead of the expand margin (see expand_margin_bottom), as increasing the expand margin does not increase the size of the clickable area for Controls.

Method Descriptions

int **get_border_width**(margin: Side) const 🔗

Returns the specified Side's border width.

---

int **get_border_width_min**() const 🔗

Returns the smallest border width out of all four borders.

---

int **get_corner_radius**(corner: Corner) const 🔗

Returns the given `corner`'s radius. See Corner for possible values.

---

float **get_expand_margin**(margin: Side) const 🔗

Returns the size of the specified Side's expand margin.

---

void **set_border_width**(margin: Side, width: int) 🔗

Sets the specified Side's border width to `width` pixels.

void **set_border_width_all**(width: int) 🔗

Sets the border width to `width` pixels for all sides.

void **set_corner_radius**(corner: Corner, radius: int) 🔗

Sets the corner radius to `radius` pixels for the given `corner`. See Corner for possible values.

void **set_corner_radius_all**(radius: int) 🔗

Sets the corner radius to `radius` pixels for all corners.

void **set_expand_margin**(margin: Side, size: float) 🔗

Sets the expand margin to `size` pixels for the specified Side.

void **set_expand_margin_all**(size: float) 🔗

Sets the expand margin to `size` pixels for all sides.

StyleBoxLine

Inherits: StyleBox **<** Resource **<** RefCounted **<** Object

A StyleBox that displays a single line of a given color and thickness.

Description

A StyleBox that displays a single line of a given color and thickness. The line can be either horizontal or vertical. Useful for separators.

Properties

Color	color	`Color(0, 0, 0, 1)`
float	grow_begin	`1.0`
float	grow_end	`1.0`
int	thickness	`1`
bool	vertical	`false`

Property Descriptions

Color **color** = `Color(0, 0, 0, 1)`

- void **set_color**(value: Color)
- Color **get_color**()

The line's color.

---

float **grow_begin** = `1.0`

- void **set_grow_begin**(value: float)
- float **get_grow_begin**()

The number of pixels the line will extend before the **StyleBoxLine**'s bounds. If set to a negative value, the line will begin inside the **StyleBoxLine**'s bounds.

---

float **grow_end** = `1.0`

- void **set_grow_end**(value: float)
- float **get_grow_end**()

The number of pixels the line will extend past the **StyleBoxLine**'s bounds. If set to a negative value, the line will end inside the **StyleBoxLine**'s bounds.

---

int **thickness** = `1`

- void **set_thickness**(value: int)
- int **get_thickness**()

The line's thickness in pixels.

---

bool **vertical** = `false`

- void **set_vertical**(value: bool)
- bool **is_vertical**()

If `true`, the line will be vertical. If `false`, the line will be horizontal.

StyleBoxTexture

Inherits: StyleBox **<** Resource **<** RefCounted **<** Object

A texture-based nine-patch StyleBox.

Description

A texture-based nine-patch StyleBox, in a way similar to NinePatchRect. This stylebox performs a 3×3 scaling of a texture, where only the center cell is fully stretched. This makes it possible to design bordered styles regardless of the stylebox's size.

Properties

AxisStretchMode	axis_stretch_horizontal	`0`
AxisStretchMode	axis_stretch_vertical	`0`
bool	draw_center	`true`
float	expand_margin_bottom	`0.0`
float	expand_margin_left	`0.0`
float	expand_margin_right	`0.0`

float	expand_margin_top	`0.0`
Color	modulate_color	`Color(1, 1, 1, 1)`
Rect2	region_rect	`Rect2(0, 0, 0, 0)`
Texture2D	texture	
float	texture_margin_bottom	`0.0`
float	texture_margin_left	`0.0`
float	texture_margin_right	`0.0`
float	texture_margin_top	`0.0`

Methods

float	get_expand_margin(margin: Side) const
float	get_texture_margin(margin: Side) const
void	set_expand_margin(margin: Side, size: float)

void	set_expand_margin_all(size: float)
void	set_texture_margin(margin: Side, size: float)
void	set_texture_margin_all(size: float)

Enumerations

enum **AxisStretchMode**: 🔗

AxisStretchMode **AXIS_STRETCH_MODE_STRETCH** = `0`

Stretch the stylebox's texture. This results in visible distortion unless the texture size matches the stylebox's size perfectly.

AxisStretchMode **AXIS_STRETCH_MODE_TILE** = `1`

Repeats the stylebox's texture to match the stylebox's size according to the nine-patch system.

AxisStretchMode **AXIS_STRETCH_MODE_TILE_FIT** = `2`

Repeats the stylebox's texture to match the stylebox's size according to the nine-patch system. Unlike AXIS_STRETCH_MODE_TILE, the texture may be slightly stretched to make the nine-patch texture tile seamlessly.

Property Descriptions

AxisStretchMode **axis_stretch_horizontal** = `0` 🔗

- void **set_h_axis_stretch_mode**(value: AxisStretchMode)
- AxisStretchMode **get_h_axis_stretch_mode**()

Controls how the stylebox's texture will be stretched or tiled horizontally. See AxisStretchMode for possible values.

AxisStretchMode **axis_stretch_vertical** = `0` 🔗

- void **set_v_axis_stretch_mode**(value: AxisStretchMode)
- AxisStretchMode **get_v_axis_stretch_mode**()

Controls how the stylebox's texture will be stretched or tiled vertically. See AxisStretchMode for possible values.

bool **draw_center** = `true` 🔗

- void **set_draw_center**(value: bool)
- bool **is_draw_center_enabled**()

If `true`, the nine-patch texture's center tile will be drawn.

float **expand_margin_bottom** = `0.0` 🔗

- void **set_expand_margin**(margin: Side, size: float)
- float **get_expand_margin**(margin: Side) const

Expands the bottom margin of this style box when drawing, causing it to be drawn larger than requested.

float **expand_margin_left** = `0.0` 🔗

- void **set_expand_margin**(margin: Side, size: float)
- float **get_expand_margin**(margin: Side) const

Expands the left margin of this style box when drawing, causing it to be drawn larger than requested.

---

float **expand_margin_right** = `0.0` 🔗

- void **set_expand_margin**(margin: Side, size: float)
- float **get_expand_margin**(margin: Side) const

Expands the right margin of this style box when drawing, causing it to be drawn larger than requested.

---

float **expand_margin_top** = `0.0` 🔗

- void **set_expand_margin**(margin: Side, size: float)
- float **get_expand_margin**(margin: Side) const

Expands the top margin of this style box when drawing, causing it to be drawn larger than requested.

Color **modulate_color** = `Color(1, 1, 1, 1)` 🔗

- void **set_modulate**(value: Color)
- Color **get_modulate**()

Modulates the color of the texture when this style box is drawn.

Rect2 **region_rect** = `Rect2(0, 0, 0, 0)` 🔗

- void **set_region_rect**(value: Rect2)
- Rect2 **get_region_rect**()

Species a sub-region of the texture to use.

This is equivalent to first wrapping the texture in an AtlasTexture with the same region.

If empty (`Rect2(0, 0, 0, 0)`), the whole texture will be used.

Texture2D **texture** 🔗

- void **set_texture**(value: Texture2D)
- Texture2D **get_texture**()

The texture to use when drawing this style box.

---

float **texture_margin_bottom** = `0.0` 🔗

- void **set_texture_margin**(margin: Side, size: float)
- float **get_texture_margin**(margin: Side) const

Increases the bottom margin of the 3×3 texture box.

A higher value means more of the source texture is considered to be part of the bottom border of the 3×3 box.

This is also the value used as fallback for StyleBox.content_margin_bottom if it is negative.

---

float **texture_margin_left** = `0.0` 🔗

- void **set_texture_margin**(margin: Side, size: float)
- float **get_texture_margin**(margin: Side) const

Increases the left margin of the 3×3 texture box.

A higher value means more of the source texture is considered to be part of the left border of the 3×3 box.

This is also the value used as fallback for StyleBox.content_margin_left if it is negative.

---

float **texture_margin_right** = `0.0` 🔗

- void **set_texture_margin**(margin: Side, size: float)
- float **get_texture_margin**(margin: Side) const

Increases the right margin of the 3×3 texture box.

A higher value means more of the source texture is considered to be part of the right border of the 3×3 box.

This is also the value used as fallback for StyleBox.content_margin_right if it is negative.

---

float **texture_margin_top** = `0.0` 🔗

- void **set_texture_margin**(margin: Side, size: float)
- float **get_texture_margin**(margin: Side) const

Increases the top margin of the 3×3 texture box.

A higher value means more of the source texture is considered to be part of the top border of the 3×3 box.

This is also the value used as fallback for StyleBox.content_margin_top if it is negative.

Method Descriptions

float **get_expand_margin**(margin: Side) const

Returns the expand margin size of the specified Side.

---

float **get_texture_margin**(margin: Side) const

Returns the margin size of the specified Side.

---

void **set_expand_margin**(margin: Side, size: float)

Sets the expand margin to `size` pixels for the specified Side.

---

void **set_expand_margin_all**(size: float)

Sets the expand margin to `size` pixels for all sides.

---

void **set_texture_margin**(margin: Side, size: float)

Sets the margin to `size` pixels for the specified Side.

---

void **set_texture_margin_all**(size: float)

Sets the margin to `size` pixels for all sides.

SyntaxHighlighter

Inherits: Resource **<** RefCounted **<** Object

Inherited By: CodeHighlighter, EditorSyntaxHighlighter

Base class for syntax highlighters. Provides syntax highlighting data to a TextEdit.

Description

Base class for syntax highlighters. Provides syntax highlighting data to a TextEdit. The associated TextEdit will call into the **SyntaxHighlighter** on an as-needed basis.

Note: A **SyntaxHighlighter** instance should not be used across multiple TextEdit nodes.

Methods

void	_clear_highlighting_cache() virtual
Dictionary	_get_line_syntax_highlighting(line: int) virtual const
void	_update_cache() virtual
void	clear_highlighting_cache()

Dictionary	get_line_syntax_highlighting(line: int)
TextEdit	get_text_edit() const
void	update_cache()

Method Descriptions

void **_clear_highlighting_cache**() virtual

Virtual method which can be overridden to clear any local caches.

Dictionary **_get_line_syntax_highlighting**(line: int) virtual const

Virtual method which can be overridden to return syntax highlighting data.

See get_line_syntax_highlighting for more details.

---

void **_update_cache**() virtual

Virtual method which can be overridden to update any local caches.

---

void **clear_highlighting_cache**()

Clears all cached syntax highlighting data.

Then calls overridable method _clear_highlighting_cache.

Dictionary **get_line_syntax_highlighting**(line: int) 🔗

Returns syntax highlighting data for a single line. If the line is not cached, calls _get_line_syntax_highlighting to calculate the data.

The return Dictionary is column number to Dictionary. The column number notes the start of a region, the region will end if another region is found, or at the end of the line. The nested Dictionary contains the data for that region, currently only the key "color" is supported.

Example return:

```
var color_map = {
    0: {
        "color": Color(1, 0, 0)
    },
    5: {
        "color": Color(0, 1, 0)
    }
}
```

This will color columns 0-4 red, and columns 5-eol in green.

TextEdit **get_text_edit**() const 🔗

Returns the associated TextEdit node.

---

void **update_cache**() 🔗

Clears then updates the **SyntaxHighlighter** caches. Override _update_cache for a callback.

Note: This is called automatically when the associated TextEdit node, updates its own cache.

SystemFont

Inherits: Font **<** Resource **<** RefCounted **<** Object

A font loaded from a system font. Falls back to a default theme font if not implemented on the host OS.

Description

SystemFont loads a font from a system font with the first matching name from font_names.

It will attempt to match font style, but it's not guaranteed.

The returned font might be part of a font collection or be a variable font with OpenType "weight", "width" and/or "italic" features set.

You can create FontVariation of the system font for precise control over its features.

Note: This class is implemented on iOS, Linux, macOS and Windows, on other platforms it will fallback to default theme font.

Properties

bool	allow_system_fallback	`true`
FontAntialiasing	antialiasing	`1`
bool	disable_embedded_bitmaps	`true`
bool	font_italic	`false`

PackedStringArray	font_names	`PackedStringArray()`
int	font_stretch	`100`
int	font_weight	`400`
bool	force_autohinter	`false`
bool	generate_mipmaps	`false`
Hinting	hinting	`1`
int	msdf_pixel_range	`16`
int	msdf_size	`48`
bool	multichannel_signed_distance_field	`false`
float	oversampling	`0.0`
SubpixelPositioning	subpixel_positioning	`1`

Property Descriptions

bool **allow_system_fallback** = `true` 🔗

- void **set_allow_system_fallback**(value: bool)
- bool **is_allow_system_fallback**()

If set to `true`, system fonts can be automatically used as fallbacks.

FontAntialiasing **antialiasing** = `1` 🔗

- void **set_antialiasing**(value: FontAntialiasing)
- FontAntialiasing **get_antialiasing**()

Font anti-aliasing mode.

---

bool **disable_embedded_bitmaps** = `true` 🔗

- void **set_disable_embedded_bitmaps**(value: bool)
- bool **get_disable_embedded_bitmaps**()

If set to `true`, embedded font bitmap loading is disabled (bitmap-only and color fonts ignore this property).

---

bool **font_italic** = `false` 🔗

- void **set_font_italic**(value: bool)
- bool **get_font_italic**()

If set to `true`, italic or oblique font is preferred.

PackedStringArray **font_names** = `PackedStringArray()` 🔗

- void **set_font_names**(value: PackedStringArray)
- PackedStringArray **get_font_names**()

Array of font family names to search, first matching font found is used.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedStringArray for more details.

---

int **font_stretch** = `100` 🔗

- void **set_font_stretch**(value: int)

- int **get_font_stretch**()

Preferred font stretch amount, compared to a normal width. A percentage value between `50%` and `200%`.

---

int **font_weight** = `400`

- void **set_font_weight**(value: int)
- int **get_font_weight**()

Preferred weight (boldness) of the font. A value in the `100...999` range, normal font weight is `400`, bold font weight is `700`.

---

bool **force_autohinter** = `false`

- void **set_force_autohinter**(value: bool)
- bool **is_force_autohinter**()

If set to `true`, auto-hinting is supported and preferred over font built-in hinting.

---

bool **generate_mipmaps** = `false`

- void **set_generate_mipmaps**(value: bool)
- bool **get_generate_mipmaps**()

If set to `true`, generate mipmaps for the font textures.

Hinting **hinting** = `1`

- void **set_hinting**(value: Hinting)
- Hinting **get_hinting**()

Font hinting mode.

---

int **msdf_pixel_range** = `16`

- void **set_msdf_pixel_range**(value: int)
- int **get_msdf_pixel_range**()

The width of the range around the shape between the minimum and maximum representable signed distance. If using font outlines, msdf_pixel_range must be set to at least *twice* the size of the largest font outline. The default msdf_pixel_range value of `16` allows outline sizes up to `8` to look correct.

---

int **msdf_size** = `48`

- void **set_msdf_size**(value: int)
- int **get_msdf_size**()

Source font size used to generate MSDF textures. Higher values allow for more precision, but are slower to render and require more memory. Only increase this value if you notice a visible lack of precision in glyph rendering.

---

bool **multichannel_signed_distance_field** = `false`

- void **set_multichannel_signed_distance_field**(value: bool)
- bool **is_multichannel_signed_distance_field**()

If set to `true`, glyphs of all sizes are rendered using single multichannel signed distance field generated from the dynamic font vector data.

---

float **oversampling** = `0.0`

- void **set_oversampling**(value: float)
- float **get_oversampling**()

Font oversampling factor, if set to `0.0` global oversampling factor is used instead.

SubpixelPositioning **subpixel_positioning** = `1`

- void **set_subpixel_positioning**(value: SubpixelPositioning)
- SubpixelPositioning **get_subpixel_positioning**()

Font glyph subpixel positioning mode. Subpixel positioning provides shaper text and better kerning for smaller font sizes, at the cost of memory usage and font rasterization speed. Use TextServer.SUBPIXEL_POSITIONING_AUTO to automatically enable it based on the font size.

TextMesh

Inherits: PrimitiveMesh **<** Mesh **<** Resource **<** RefCounted **<** Object

Generate an PrimitiveMesh from the text.

Description

Generate an PrimitiveMesh from the text.

TextMesh can be generated only when using dynamic fonts with vector glyph contours. Bitmap fonts (including bitmap data in the TrueType/OpenType containers, like color emoji fonts) are not supported.

The UV layout is arranged in 4 horizontal strips, top to bottom: 40% of the height for the front face, 40% for the back face, 10% for the outer edges and 10% for the inner edges.

Tutorials

- 3D text

Properties

AutowrapMode	autowrap_mode	`0`
float	curve_step	`0.5`
float	depth	`0.05`
Font	font	

int	font_size	`16`
HorizontalAlignment	horizontal_alignment	`1`
BitField[JustificationFlag]	justification_flags	`163`
String	language	`""`
float	line_spacing	`0.0`
Vector2	offset	`Vector2(0, 0)`
float	pixel_size	`0.01`
StructuredTextParser	structured_text_bidi_override	`0`
Array	structured_text_bidi_override_options	`[]`
String	text	`""`
Direction	text_direction	`0`
bool	uppercase	`false`
VerticalAlignment	vertical_alignment	`1`
float	width	`500.0`

Property Descriptions

AutowrapMode **autowrap_mode** = `0` 🔗

- void **set_autowrap_mode**(value: AutowrapMode)
- AutowrapMode **get_autowrap_mode**()

If set to something other than TextServer.AUTOWRAP_OFF, the text gets wrapped inside the node's bounding rectangle. If you resize the node, it will change its height automatically to show all the text. To see how each mode behaves, see AutowrapMode.

---

float **curve_step** = `0.5` 🔗

- void **set_curve_step**(value: float)
- float **get_curve_step**()

Step (in pixels) used to approximate Bézier curves.

---

float **depth** = `0.05` 🔗

- void **set_depth**(value: float)
- float **get_depth**()

Depths of the mesh, if set to `0.0` only front surface, is generated, and UV layout is changed to use full texture for the front face only.

Font **font** 🔗

- void **set_font**(value: Font)
- Font **get_font**()

Font configuration used to display text.

---

int **font_size** = `16` 🔗

- void **set_font_size**(value: int)
- int **get_font_size**()

Font size of the **TextMesh**'s text.

HorizontalAlignment **horizontal_alignment** = `1` 🔗

- void **set_horizontal_alignment**(value: HorizontalAlignment)
- HorizontalAlignment **get_horizontal_alignment**()

Controls the text's horizontal alignment. Supports left, center, right, and fill, or justify. Set it to one of the HorizontalAlignment constants.

BitField[JustificationFlag] **justification_flags** = `163` 🔗

- void **set_justification_flags**(value: BitField[JustificationFlag])
- BitField[JustificationFlag] **get_justification_flags**()

Line fill alignment rules. For more info see JustificationFlag.

String **language** = `""` 🔗

- void **set_language**(value: String)
- String **get_language**()

Language code used for text shaping algorithms, if left empty current locale is used instead.

---

float **line_spacing** = `0.0` 🔗

- void **set_line_spacing**(value: float)
- float **get_line_spacing**()

Vertical space between lines in multiline **TextMesh**.

Vector2 **offset** = `Vector2(0, 0)`

- void **set_offset**(value: Vector2)
- Vector2 **get_offset**()

The text drawing offset (in pixels).

---

float **pixel_size** = `0.01`

- void **set_pixel_size**(value: float)
- float **get_pixel_size**()

The size of one pixel's width on the text to scale it in 3D.

StructuredTextParser **structured_text_bidi_override** = `0`

- void **set_structured_text_bidi_override**(value: StructuredTextParser)
- StructuredTextParser **get_structured_text_bidi_override**()

Set BiDi algorithm override for the structured text.

Array **structured_text_bidi_override_options** = `[]`

- void **set_structured_text_bidi_override_options**(value: Array)
- Array **get_structured_text_bidi_override_options**()

Set additional options for BiDi override.

String **text** = `""`

- void **set_text**(value: String)
- String **get_text**()

The text to generate mesh from.

Note: Due to being a Resource, it doesn't follow the rules of Node.auto_translate_mode. If disabling translation is desired, it should be done manually with Object.set_message_translation.

Direction **text_direction** = `0`

- void **set_text_direction**(value: Direction)
- Direction **get_text_direction**()

Base text writing direction.

bool **uppercase** = `false`

- void **set_uppercase**(value: bool)
- bool **is_uppercase**()

If `true`, all the text displays as UPPERCASE.

VerticalAlignment **vertical_alignment** = `1`

- void **set_vertical_alignment**(value: VerticalAlignment)
- VerticalAlignment **get_vertical_alignment**()

Controls the text's vertical alignment. Supports top, center, bottom. Set it to one of the VerticalAlignment constants.

float **width** = `500.0`

- void **set_width**(value: float)
- float **get_width**()

Text width (in pixels), used for fill alignment.

Texture

Inherits: Resource **<** RefCounted **<** Object

Inherited By: Texture2D, Texture3D, TextureLayered

Base class for all texture types.

Description

Texture is the base class for all texture types. Common texture types are Texture2D and ImageTexture. See also Image.

Texture2D

Inherits: Texture **<** Resource **<** RefCounted **<** Object

Inherited By: AnimatedTexture, AtlasTexture, CameraTexture, CanvasTexture, CompressedTexture2D, CurveTexture, CurveXYZTexture, GradientTexture1D, GradientTexture2D, ImageTexture, MeshTexture, NoiseTexture2D, PlaceholderTexture2D, PortableCompressedTexture2D, Texture2DRD, ViewportTexture

Texture for 2D and 3D.

Description

A texture works by registering an image in the video hardware, which then can be used in 3D models or 2D Sprite2D or GUI Control.

Textures are often created by loading them from a file. See @GDScript.load.

Texture2D is a base for other resources. It cannot be used directly.

Note: The maximum texture size is 16384×16384 pixels due to graphics hardware limitations. Larger textures may fail to import.

Methods

void	_draw(to_canvas_item: RID, pos: Vector2, modulate: Color, transpose: bool) virtual const

void	_draw_rect(to_canvas_item: RID, rect: Rect2, tile: bool, modulate: Color, transpose: bool) virtual const
void	_draw_rect_region(to_canvas_item: RID, rect: Rect2, src_rect: Rect2, modulate: Color, transpose: bool, clip_uv: bool) virtual const
int	_get_height() virtual const
int	_get_width() virtual const
bool	_has_alpha() virtual const
bool	_is_pixel_opaque(x: int, y: int) virtual const
Resource	create_placeholder() const
void	draw(canvas_item: RID, position: Vector2, modulate: Color = Color(1, 1, 1, 1), transpose: bool = false) const
void	draw_rect(canvas_item: RID, rect: Rect2, tile: bool, modulate: Color = Color(1, 1, 1, 1), transpose: bool = false) const
void	draw_rect_region(canvas_item: RID, rect: Rect2, src_rect: Rect2, modulate: Color = Color(1, 1, 1, 1), transpose: bool = false, clip_uv: bool = true) const

int	get_height() const
Image	get_image() const
Vector2	get_size() const
int	get_width() const
bool	has_alpha() const

Method Descriptions

void **_draw**(to_canvas_item: RID, pos: Vector2, modulate: Color, transpose: bool) virtual const 🔗

Called when the entire **Texture2D** is requested to be drawn over a CanvasItem, with the top-left offset specified in `pos`. `modulate` specifies a multiplier for the colors being drawn, while `transpose` specifies whether drawing should be performed in column-major order instead of row-major order (resulting in 90-degree clockwise rotation).

Note: This is only used in 2D rendering, not 3D.

void **_draw_rect**(to_canvas_item: RID, rect: Rect2, tile: bool, modulate: Color, transpose: bool) virtual const 🔗

Called when the **Texture2D** is requested to be drawn onto CanvasItem's specified `rect`. `modulate` specifies a multiplier for the colors being drawn, while `transpose` specifies whether drawing should be performed in column-major order instead of row-major order (resulting in 90-degree clockwise rotation).

Note: This is only used in 2D rendering, not 3D.

void **_draw_rect_region**(to_canvas_item: RID, rect: Rect2, src_rect: Rect2, modulate: Color, transpose: bool, clip_uv: bool) virtual const 🔗

Called when a part of the **Texture2D** specified by `src_rect`'s coordinates is requested to be drawn onto CanvasItem's specified `rect`. `modulate` specifies a multiplier for the colors being drawn, while `transpose` specifies whether drawing should be performed in column-major order instead of row-major order (resulting in 90-degree clockwise rotation).

Note: This is only used in 2D rendering, not 3D.

int **_get_height**() virtual const 🔗

Called when the **Texture2D**'s height is queried.

int **_get_width**() virtual const 🔗

Called when the **Texture2D**'s width is queried.

bool **_has_alpha**() virtual const 🔗

Called when the presence of an alpha channel in the **Texture2D** is queried.

bool **_is_pixel_opaque**(x: int, y: int) virtual const

Called when a pixel's opaque state in the **Texture2D** is queried at the specified `(x, y)` position.

Resource **create_placeholder**() const

Creates a placeholder version of this resource (PlaceholderTexture2D).

void **draw**(canvas_item: RID, position: Vector2, modulate: Color = Color(1, 1, 1, 1), transpose: bool = false) const

Draws the texture using a CanvasItem with the RenderingServer API at the specified `position`.

void **draw_rect**(canvas_item: RID, rect: Rect2, tile: bool, modulate: Color = Color(1, 1, 1, 1), transpose: bool = false) const

Draws the texture using a CanvasItem with the RenderingServer API.

void **draw_rect_region**(canvas_item: RID, rect: Rect2, src_rect: Rect2, modulate: Color = Color(1, 1, 1, 1), transpose: bool = false, clip_uv: bool = true) const

Draws a part of the texture using a CanvasItem with the RenderingServer API.

int **get_height**() const

Returns the texture height in pixels.

Image **get_image**() const

Returns an Image that is a copy of data from this **Texture2D** (a new Image is created each time). Images can be accessed and manipulated directly.

Note: This will return `null` if this **Texture2D** is invalid.

Note: This will fetch the texture data from the GPU, which might cause performance problems when overused.

Vector2 **get_size**() const

Returns the texture size in pixels.

int **get_width**() const

Returns the texture width in pixels.

bool **has_alpha**() const

Returns `true` if this **Texture2D** has an alpha channel.

Texture2DArray

Inherits: ImageTextureLayered **<** TextureLayered **<** Texture **<** Resource **<** RefCounted **<** Object

A single texture resource which consists of multiple, separate images. Each image has the same dimensions and number of mipmap levels.

Description

A Texture2DArray is different from a Texture3D: The Texture2DArray does not support trilinear interpolation between the Images, i.e. no blending. See also Cubemap and CubemapArray, which are texture arrays with specialized cubemap functions.

A Texture2DArray is also different from an AtlasTexture: In a Texture2DArray, all images are treated separately. In an atlas, the regions (i.e. the single images) can be of different sizes. Furthermore, you usually need to add a padding around the regions, to prevent accidental UV mapping to more than one region. The same goes for mipmapping: Mipmap chains are handled separately for each layer. In an atlas, the slicing has to be done manually in the fragment shader.

To create such a texture file yourself, reimport your image files using the Godot Editor import presets.

Methods

Resource	create_placeholder() const

Method Descriptions

Resource **create_placeholder**() const 🔗

Creates a placeholder version of this resource (PlaceholderTexture2DArray).

Texture2DArrayRD

Inherits: TextureLayeredRD **<** TextureLayered **<** Texture **<** Resource **<** RefCounted **<** Object

Texture Array for 2D that is bound to a texture created on the RenderingDevice.

Description

This texture array class allows you to use a 2D array texture created directly on the RenderingDevice as a texture for materials, meshes, etc.

Texture2DRD

Inherits: Texture2D **<** Texture **<** Resource **<** RefCounted **<** Object

Texture for 2D that is bound to a texture created on the RenderingDevice.

Description

This texture class allows you to use a 2D texture created directly on the RenderingDevice as a texture for materials, meshes, etc.

Properties

bool	resource_local_to_scene	`false` (overrides Resource)
RID	texture_rd_rid	

Property Descriptions

RID **texture_rd_rid** 🔗

- void **set_texture_rd_rid**(value: RID)
- RID **get_texture_rd_rid**()

The RID of the texture object created on the RenderingDevice.

Texture3D

Inherits: Texture **<** Resource **<** RefCounted **<** Object

Inherited By: CompressedTexture3D, ImageTexture3D, NoiseTexture3D, PlaceholderTexture3D, Texture3DRD

Base class for 3-dimensional textures.

Description

Base class for ImageTexture3D and CompressedTexture3D. Cannot be used directly, but contains all the functions necessary for accessing the derived resource types. **Texture3D** is the base class for all 3-dimensional texture types. See also TextureLayered.

All images need to have the same width, height and number of mipmap levels.

To create such a texture file yourself, reimport your image files using the Godot Editor import presets.

Methods

Array[Image]	_get_data() virtual const
int	_get_depth() virtual const
Format	_get_format() virtual const

int	_get_height() virtual const
int	_get_width() virtual const
bool	_has_mipmaps() virtual const
Resource	create_placeholder() const
Array[Image]	get_data() const
int	get_depth() const
Format	get_format() const
int	get_height() const
int	get_width() const
bool	has_mipmaps() const

Method Descriptions

Array[Image] **_get_data**() virtual const 🔗

Called when the **Texture3D**'s data is queried.

---

int **_get_depth**() virtual const

Called when the **Texture3D**'s depth is queried.

Format **_get_format**() virtual const

Called when the **Texture3D**'s format is queried.

---

int **_get_height**() virtual const

Called when the **Texture3D**'s height is queried.

---

int **_get_width**() virtual const

Called when the **Texture3D**'s width is queried.

---

bool **_has_mipmaps**() virtual const

Called when the presence of mipmaps in the **Texture3D** is queried.

Resource **create_placeholder**() const

Creates a placeholder version of this resource (PlaceholderTexture3D).

Array[Image] **get_data**() const

Returns the **Texture3D**'s data as an array of Images. Each Image represents a *slice* of the **Texture3D**, with different slices mapping to different depth (Z axis) levels.

---

int **get_depth**() const

Returns the **Texture3D**'s depth in pixels. Depth is typically represented by the Z axis (a dimension not present in Texture2D).

Format **get_format**() const

Returns the current format being used by this texture. See Format for details.

---

int **get_height**() const

Returns the **Texture3D**'s height in pixels. Width is typically represented by the Y axis.

---

int **get_width**() const

Returns the **Texture3D**'s width in pixels. Width is typically represented by the X axis.

---

bool **has_mipmaps**() const

Returns `true` if the **Texture3D** has generated mipmaps.

Texture3DRD

Inherits: Texture3D **<** Texture **<** Resource **<** RefCounted **<** Object

Texture for 3D that is bound to a texture created on the RenderingDevice.

Description

This texture class allows you to use a 3D texture created directly on the RenderingDevice as a texture for materials, meshes, etc.

Properties

RID	texture_rd_rid

Property Descriptions

RID **texture_rd_rid**

- void **set_texture_rd_rid**(value: RID)
- RID **get_texture_rd_rid**()

The RID of the texture object created on the RenderingDevice.

TextureCubemapArrayRD

Inherits: TextureLayeredRD **<** TextureLayered **<** Texture **<** Resource **<** RefCounted **<** Object

Texture Array for Cubemaps that is bound to a texture created on the RenderingDevice.

Description

This texture class allows you to use a cubemap array texture created directly on the RenderingDevice as a texture for materials, meshes, etc.

TextureCubemapRD

Inherits: TextureLayeredRD **<** TextureLayered **<** Texture **<** Resource **<** RefCounted **<** Object

Texture for Cubemap that is bound to a texture created on the RenderingDevice.

Description

This texture class allows you to use a cubemap texture created directly on the RenderingDevice as a texture for materials, meshes, etc.

TextureLayered

Inherits: Texture **<** Resource **<** RefCounted **<** Object

Inherited By: CompressedTextureLayered, ImageTextureLayered, PlaceholderTextureLayered, TextureLayeredRD

Base class for texture types which contain the data of multiple Images. Each image is of the same size and format.

Description

Base class for ImageTextureLayered and CompressedTextureLayered. Cannot be used directly, but contains all the functions necessary for accessing the derived resource types. See also Texture3D.

Data is set on a per-layer basis. For Texture2DArrays, the layer specifies the array layer.

All images need to have the same width, height and number of mipmap levels.

A **TextureLayered** can be loaded with ResourceLoader.load.

Internally, Godot maps these files to their respective counterparts in the target rendering driver (Vulkan, OpenGL3).

Methods

Format	_get_format() virtual const

int	_get_height() virtual const
Image	_get_layer_data(layer_index: int) virtual const
int	_get_layered_type() virtual const
int	_get_layers() virtual const
int	_get_width() virtual const
bool	_has_mipmaps() virtual const
Format	get_format() const
int	get_height() const
Image	get_layer_data(layer: int) const
LayeredType	get_layered_type() const
int	get_layers() const
int	get_width() const
bool	has_mipmaps() const

Enumerations

enum **LayeredType**: 🔗

LayeredType **LAYERED_TYPE_2D_ARRAY** = `0`

Texture is a generic Texture2DArray.

LayeredType **LAYERED_TYPE_CUBEMAP** = `1`

Texture is a Cubemap, with each side in its own layer (6 in total).

LayeredType **LAYERED_TYPE_CUBEMAP_ARRAY** = `2`

Texture is a CubemapArray, with each cubemap being made of 6 layers.

Method Descriptions

Format **_get_format**() virtual const 🔗

Called when the **TextureLayered**'s format is queried.

int **_get_height**() virtual const 🔗

Called when the **TextureLayered**'s height is queried.

Image **_get_layer_data**(layer_index: int) virtual const 🔗

Called when the data for a layer in the **TextureLayered** is queried.

---

int **_get_layered_type**() virtual const

Called when the layers' type in the **TextureLayered** is queried.

---

int **_get_layers**() virtual const

Called when the number of layers in the **TextureLayered** is queried.

---

int **_get_width**() virtual const

Called when the **TextureLayered**'s width queried.

---

bool **_has_mipmaps**() virtual const

Called when the presence of mipmaps in the **TextureLayered** is queried.

Format **get_format**() const

Returns the current format being used by this texture. See Format for details.

---

int **get_height**() const

Returns the height of the texture in pixels. Height is typically represented by the Y axis.

Image **get_layer_data**(layer: int) const

Returns an Image resource with the data from specified `layer`.

LayeredType **get_layered_type**() const

Returns the **TextureLayered**'s type. The type determines how the data is accessed, with cubemaps having special types.

int **get_layers**() const

Returns the number of referenced Images.

int **get_width**() const

Returns the width of the texture in pixels. Width is typically represented by the X axis.

bool **has_mipmaps**() const

Returns `true` if the layers have generated mipmaps.

TextureLayeredRD

Inherits: TextureLayered **<** Texture **<** Resource **<** RefCounted **<** Object

Inherited By: Texture2DArrayRD, TextureCubemapArrayRD, TextureCubemapRD

Abstract base class for layered texture RD types.

Description

Base class for Texture2DArrayRD, TextureCubemapRD and TextureCubemapArrayRD. Cannot be used directly, but contains all the functions necessary for accessing the derived resource types.

Properties

RID	texture_rd_rid

Property Descriptions

RID **texture_rd_rid** 🔗

- void **set_texture_rd_rid**(value: RID)
- RID **get_texture_rd_rid**()

The RID of the texture object created on the RenderingDevice.

Theme

Inherits: Resource **<** RefCounted **<** Object

A resource used for styling/skinning Controls and Windows.

Description

A resource used for styling/skinning Control and Window nodes. While individual controls can be styled using their local theme overrides (see Control.add_theme_color_override), theme resources allow you to store and apply the same settings across all controls sharing the same type (e.g. style all Buttons the same). One theme resource can be used for the entire project, but you can also set a separate theme resource to a branch of control nodes. A theme resource assigned to a control applies to the control itself, as well as all of its direct and indirect children (as long as a chain of controls is uninterrupted).

Use ProjectSettings.gui/theme/custom to set up a project-scope theme that will be available to every control in your project.

Use Control.theme of any control node to set up a theme that will be available to that control and all of its direct and indirect children.

Tutorials

- GUI skinning
- Using the theme editor

Properties

float	default_base_scale	`0.0`
Font	default_font	
int	default_font_size	`-1`

Methods

void	add_type(theme_type: StringName)
void	clear()
void	clear_color(name: StringName, theme_type: StringName)
void	clear_constant(name: StringName, theme_type: StringName)
void	clear_font(name: StringName, theme_type: StringName)
void	clear_font_size(name: StringName, theme_type: StringName)

void	clear_icon(name: StringName, theme_type: StringName)
void	clear_stylebox(name: StringName, theme_type: StringName)
void	clear_theme_item(data_type: DataType, name: StringName, theme_type: StringName)
void	clear_type_variation(theme_type: StringName)
Color	get_color(name: StringName, theme_type: StringName) const
PackedStringArray	get_color_list(theme_type: String) const
PackedStringArray	get_color_type_list() const
int	get_constant(name: StringName, theme_type: StringName) const
PackedStringArray	get_constant_list(theme_type: String) const
PackedStringArray	get_constant_type_list() const
Font	get_font(name: StringName, theme_type: StringName) const

PackedStringArray	get_font_list(theme_type: String) const
int	get_font_size(name: StringName, theme_type: StringName) const
PackedStringArray	get_font_size_list(theme_type: String) const
PackedStringArray	get_font_size_type_list() const
PackedStringArray	get_font_type_list() const
Texture2D	get_icon(name: StringName, theme_type: StringName) const
PackedStringArray	get_icon_list(theme_type: String) const
PackedStringArray	get_icon_type_list() const
StyleBox	get_stylebox(name: StringName, theme_type: StringName) const
PackedStringArray	get_stylebox_list(theme_type: String) const
PackedStringArray	get_stylebox_type_list() const

Variant	get_theme_item(data_type: DataType, name: StringName, theme_type: StringName) const
PackedStringArray	get_theme_item_list(data_type: DataType, theme_type: String) const
PackedStringArray	get_theme_item_type_list(data_type: DataType) const
PackedStringArray	get_type_list() const
StringName	get_type_variation_base(theme_type: StringName) const
PackedStringArray	get_type_variation_list(base_type: StringName) const
bool	has_color(name: StringName, theme_type: StringName) const
bool	has_constant(name: StringName, theme_type: StringName) const
bool	has_default_base_scale() const
bool	has_default_font() const

bool	has_default_font_size() const
bool	has_font(name: StringName, theme_type: StringName) const
bool	has_font_size(name: StringName, theme_type: StringName) const
bool	has_icon(name: StringName, theme_type: StringName) const
bool	has_stylebox(name: StringName, theme_type: StringName) const
bool	has_theme_item(data_type: DataType, name: StringName, theme_type: StringName) const
bool	is_type_variation(theme_type: StringName, base_type: StringName) const
void	merge_with(other: Theme)
void	remove_type(theme_type: StringName)
void	rename_color(old_name: StringName, name: StringName, theme_type: StringName)

void	rename_constant(old_name: StringName, name: StringName, theme_type: StringName)
void	rename_font(old_name: StringName, name: StringName, theme_type: StringName)
void	rename_font_size(old_name: StringName, name: StringName, theme_type: StringName)
void	rename_icon(old_name: StringName, name: StringName, theme_type: StringName)
void	rename_stylebox(old_name: StringName, name: StringName, theme_type: StringName)
void	rename_theme_item(data_type: DataType, old_name: StringName, name: StringName, theme_type: StringName)
void	set_color(name: StringName, theme_type: StringName, color: Color)
void	set_constant(name: StringName, theme_type: StringName, constant: int)
void	set_font(name: StringName, theme_type: StringName, font: Font)

void	set_font_size(name: StringName, theme_type: StringName, font_size: int)
void	set_icon(name: StringName, theme_type: StringName, texture: Texture2D)
void	set_stylebox(name: StringName, theme_type: StringName, texture: StyleBox)
void	set_theme_item(data_type: DataType, name: StringName, theme_type: StringName, value: Variant)
void	set_type_variation(theme_type: StringName, base_type: StringName)

Enumerations

enum **DataType**: 🔗

DataType **DATA_TYPE_COLOR** = `0`

Theme's Color item type.

DataType **DATA_TYPE_CONSTANT** = `1`

Theme's constant item type.

DataType **DATA_TYPE_FONT** = `2`

Theme's Font item type.

DataType **DATA_TYPE_FONT_SIZE** = `3`

Theme's font size item type.

DataType **DATA_TYPE_ICON** = `4`

Theme's icon Texture2D item type.

DataType **DATA_TYPE_STYLEBOX** = `5`

Theme's StyleBox item type.

DataType **DATA_TYPE_MAX** = `6`

Maximum value for the DataType enum.

Property Descriptions

float **default_base_scale** = `0.0` 🔗

- void **set_default_base_scale**(value: float)
- float **get_default_base_scale**()

The default base scale factor of this theme resource. Used by some controls to scale their visual properties based on the global scale factor. If this value is set to `0.0`, the global scale factor is used (see ThemeDB.fallback_base_scale).

Use has_default_base_scale to check if this value is valid.

Font **default_font** 🔗

- void **set_default_font**(value: Font)

- Font **get_default_font**()

The default font of this theme resource. Used as the default value when trying to fetch a font resource that doesn't exist in this theme or is in invalid state. If the default font is also missing or invalid, the engine fallback value is used (see ThemeDB.fallback_font).

Use has_default_font to check if this value is valid.

---

int **default_font_size** = `-1`

- void **set_default_font_size**(value: int)
- int **get_default_font_size**()

The default font size of this theme resource. Used as the default value when trying to fetch a font size value that doesn't exist in this theme or is in invalid state. If the default font size is also missing or invalid, the engine fallback value is used (see ThemeDB.fallback_font_size).

Values below `1` are invalid and can be used to unset the property. Use has_default_font_size to check if this value is valid.

Method Descriptions

void **add_type**(theme_type: StringName)

Adds an empty theme type for every valid data type.

Note: Empty types are not saved with the theme. This method only exists to perform in-memory changes to the resource. Use available `set_*` methods to add theme items.

---

void **clear**()

Removes all the theme properties defined on the theme resource.

void **clear_color**(name: StringName, theme_type: StringName)

Removes the Color property defined by `name` and `theme_type`, if it exists.

Fails if it doesn't exist. Use has_color to check for existence.

void **clear_constant**(name: StringName, theme_type: StringName)

Removes the constant property defined by `name` and `theme_type`, if it exists.

Fails if it doesn't exist. Use has_constant to check for existence.

void **clear_font**(name: StringName, theme_type: StringName)

Removes the Font property defined by `name` and `theme_type`, if it exists.

Fails if it doesn't exist. Use has_font to check for existence.

void **clear_font_size**(name: StringName, theme_type: StringName)

Removes the font size property defined by `name` and `theme_type`, if it exists.

Fails if it doesn't exist. Use has_font_size to check for existence.

---

void **clear_icon**(name: StringName, theme_type: StringName) 🔗

Removes the icon property defined by `name` and `theme_type`, if it exists.

Fails if it doesn't exist. Use has_icon to check for existence.

---

void **clear_stylebox**(name: StringName, theme_type: StringName) 🔗

Removes the StyleBox property defined by `name` and `theme_type`, if it exists.

Fails if it doesn't exist. Use has_stylebox to check for existence.

---

void **clear_theme_item**(data_type: DataType, name: StringName, theme_type: StringName) 🔗

Removes the theme property of `data_type` defined by `name` and `theme_type`, if it exists.

Fails if it doesn't exist. Use has_theme_item to check for existence.

Note: This method is analogous to calling the corresponding data type specific method, but can be used for more generalized logic.

---

void **clear_type_variation**(theme_type: StringName) 🔗

Unmarks `theme_type` as being a variation of another theme type. See set_type_variation.

Color **get_color**(name: StringName, theme_type: StringName) const 🔗

Returns the Color property defined by `name` and `theme_type`, if it exists.

Returns the default color value if the property doesn't exist. Use has_color to check for existence.

PackedStringArray **get_color_list**(theme_type: String) const 🔗

Returns a list of names for Color properties defined with `theme_type`. Use get_color_type_list to get a list of possible theme type names.

PackedStringArray **get_color_type_list**() const 🔗

Returns a list of all unique theme type names for Color properties. Use get_type_list to get a list of all unique theme types.

---

int **get_constant**(name: StringName, theme_type: StringName) const 🔗

Returns the constant property defined by `name` and `theme_type`, if it exists.

Returns `0` if the property doesn't exist. Use has_constant to check for existence.

PackedStringArray **get_constant_list**(theme_type: String) const 🔗

Returns a list of names for constant properties defined with `theme_type`. Use get_constant_type_list to get a list of possible theme type names.

PackedStringArray **get_constant_type_list**() const

Returns a list of all unique theme type names for constant properties. Use get_type_list to get a list of all unique theme types.

Font **get_font**(name: StringName, theme_type: StringName) const

Returns the Font property defined by `name` and `theme_type`, if it exists.

Returns the default theme font if the property doesn't exist and the default theme font is set up (see default_font). Use has_font to check for existence of the property and has_default_font to check for existence of the default theme font.

Returns the engine fallback font value, if neither exist (see ThemeDB.fallback_font).

PackedStringArray **get_font_list**(theme_type: String) const

Returns a list of names for Font properties defined with `theme_type`. Use get_font_type_list to get a list of possible theme type names.

---

int **get_font_size**(name: StringName, theme_type: StringName) const

Returns the font size property defined by `name` and `theme_type`, if it exists.

Returns the default theme font size if the property doesn't exist and the default theme font size is set up (see default_font_size). Use has_font_size to check for existence of the property and has_default_font_size to check for existence of the default theme font.

Returns the engine fallback font size value, if neither exist (see ThemeDB.fallback_font_size).

PackedStringArray **get_font_size_list**(theme_type: String) const 🔗

Returns a list of names for font size properties defined with `theme_type`. Use get_font_size_type_list to get a list of possible theme type names.

PackedStringArray **get_font_size_type_list**() const 🔗

Returns a list of all unique theme type names for font size properties. Use get_type_list to get a list of all unique theme types.

PackedStringArray **get_font_type_list**() const 🔗

Returns a list of all unique theme type names for Font properties. Use get_type_list to get a list of all unique theme types.

Texture2D **get_icon**(name: StringName, theme_type: StringName) const 🔗

Returns the icon property defined by `name` and `theme_type`, if it exists.

Returns the engine fallback icon value if the property doesn't exist (see ThemeDB.fallback_icon). Use has_icon to check for existence.

PackedStringArray **get_icon_list**(theme_type: String) const 🔗

Returns a list of names for icon properties defined with `theme_type`. Use get_icon_type_list to get a list of possible theme type names.

PackedStringArray **get_icon_type_list**() const 🔗

Returns a list of all unique theme type names for icon properties. Use get_type_list to get a list of all unique theme types.

StyleBox **get_stylebox**(name: StringName, theme_type: StringName) const 🔗

Returns the StyleBox property defined by `name` and `theme_type`, if it exists.

Returns the engine fallback stylebox value if the property doesn't exist (see ThemeDB.fallback_stylebox). Use has_stylebox to check for existence.

PackedStringArray **get_stylebox_list**(theme_type: String) const 🔗

Returns a list of names for StyleBox properties defined with `theme_type`. Use get_stylebox_type_list to get a list of possible theme type names.

PackedStringArray **get_stylebox_type_list**() const 🔗

Returns a list of all unique theme type names for StyleBox properties. Use get_type_list to get a list of all unique theme types.

Variant **get_theme_item**(data_type: DataType, name: StringName, theme_type: StringName) const

Returns the theme property of `data_type` defined by `name` and `theme_type`, if it exists.

Returns the engine fallback value if the property doesn't exist (see ThemeDB). Use has_theme_item to check for existence.

Note: This method is analogous to calling the corresponding data type specific method, but can be used for more generalized logic.

PackedStringArray **get_theme_item_list**(data_type: DataType, theme_type: String) const

Returns a list of names for properties of `data_type` defined with `theme_type`. Use get_theme_item_type_list to get a list of possible theme type names.

Note: This method is analogous to calling the corresponding data type specific method, but can be used for more generalized logic.

PackedStringArray **get_theme_item_type_list**(data_type: DataType) const

Returns a list of all unique theme type names for `data_type` properties. Use get_type_list to get a list of all unique theme types.

Note: This method is analogous to calling the corresponding data type specific method, but can be used for more generalized logic.

PackedStringArray **get_type_list**() const

Returns a list of all unique theme type names. Use the appropriate `get_*_type_list` method to get a list of unique theme types for a single data type.

StringName **get_type_variation_base**(theme_type: StringName) const

Returns the name of the base theme type if `theme_type` is a valid variation type. Returns an empty string otherwise.

PackedStringArray **get_type_variation_list**(base_type: StringName) const

Returns a list of all type variations for the given `base_type`.

---

bool **has_color**(name: StringName, theme_type: StringName) const

Returns `true` if the Color property defined by `name` and `theme_type` exists.

Returns `false` if it doesn't exist. Use set_color to define it.

---

bool **has_constant**(name: StringName, theme_type: StringName) const

Returns `true` if the constant property defined by `name` and `theme_type` exists.

Returns `false` if it doesn't exist. Use set_constant to define it.

---

bool **has_default_base_scale**() const

Returns `true` if default_base_scale has a valid value.

Returns `false` if it doesn't. The value must be greater than `0.0` to be considered valid.

---

bool **has_default_font**() const

Returns `true` if default_font has a valid value.

Returns `false` if it doesn't.

---

bool **has_default_font_size**() const

Returns `true` if default_font_size has a valid value.

Returns `false` if it doesn't. The value must be greater than `0` to be considered valid.

---

bool **has_font**(name: StringName, theme_type: StringName) const

Returns `true` if the Font property defined by `name` and `theme_type` exists, or if the default theme font is set up (see has_default_font).

Returns `false` if neither exist. Use set_font to define the property.

---

bool **has_font_size**(name: StringName, theme_type: StringName) const

Returns `true` if the font size property defined by `name` and `theme_type` exists, or if the default theme font size is set up (see has_default_font_size).

Returns `false` if neither exist. Use set_font_size to define the property.

bool **has_icon**(name: StringName, theme_type: StringName) const 🔗

Returns `true` if the icon property defined by `name` and `theme_type` exists.

Returns `false` if it doesn't exist. Use set_icon to define it.

bool **has_stylebox**(name: StringName, theme_type: StringName) const 🔗

Returns `true` if the StyleBox property defined by `name` and `theme_type` exists.

Returns `false` if it doesn't exist. Use set_stylebox to define it.

bool **has_theme_item**(data_type: DataType, name: StringName, theme_type: StringName) const 🔗

Returns `true` if the theme property of `data_type` defined by `name` and `theme_type` exists.

Returns `false` if it doesn't exist. Use set_theme_item to define it.

Note: This method is analogous to calling the corresponding data type specific method, but can be used for more generalized logic.

bool **is_type_variation**(theme_type: StringName, base_type: StringName) const 🔗

Returns `true` if `theme_type` is marked as a variation of `base_type`.

---

void **merge_with**(other: Theme) 🔗

Adds missing and overrides existing definitions with values from the `other` theme resource.

Note: This modifies the current theme. If you want to merge two themes together without modifying either one, create a new empty theme and merge the other two into it one after another.

---

void **remove_type**(theme_type: StringName) 🔗

Removes the theme type, gracefully discarding defined theme items. If the type is a variation, this information is also erased. If the type is a base for type variations, those variations lose their base.

---

void **rename_color**(old_name: StringName, name: StringName, theme_type: StringName) 🔗

Renames the Color property defined by `old_name` and `theme_type` to `name`, if it exists.

Fails if it doesn't exist, or if a similar property with the new name already exists. Use has_color to check for existence, and clear_color to remove the existing property.

---

void **rename_constant**(old_name: StringName, name: StringName, theme_type: StringName) 🔗

Renames the constant property defined by `old_name` and `theme_type` to `name`, if it exists.

Fails if it doesn't exist, or if a similar property with the new name already exists. Use has_constant to check for existence, and clear_constant to remove the existing property.

---

void **rename_font**(old_name: StringName, name: StringName, theme_type: StringName) 🔗

Renames the Font property defined by `old_name` and `theme_type` to `name`, if it exists.

Fails if it doesn't exist, or if a similar property with the new name already exists. Use has_font to check for existence, and clear_font to remove the existing property.

---

void **rename_font_size**(old_name: StringName, name: StringName, theme_type: StringName) 🔗

Renames the font size property defined by `old_name` and `theme_type` to `name`, if it exists.

Fails if it doesn't exist, or if a similar property with the new name already exists. Use has_font_size to check for existence, and clear_font_size to remove the existing property.

---

void **rename_icon**(old_name: StringName, name: StringName, theme_type: StringName) 🔗

Renames the icon property defined by `old_name` and `theme_type` to `name`, if it exists.

Fails if it doesn't exist, or if a similar property with the new name already exists. Use has_icon to check for existence, and clear_icon to remove the existing property.

void **rename_stylebox**(old_name: StringName, name: StringName, theme_type: StringName)

Renames the StyleBox property defined by `old_name` and `theme_type` to `name`, if it exists.

Fails if it doesn't exist, or if a similar property with the new name already exists. Use has_stylebox to check for existence, and clear_stylebox to remove the existing property.

void **rename_theme_item**(data_type: DataType, old_name: StringName, name: StringName, theme_type: StringName)

Renames the theme property of `data_type` defined by `old_name` and `theme_type` to `name`, if it exists.

Fails if it doesn't exist, or if a similar property with the new name already exists. Use has_theme_item to check for existence, and clear_theme_item to remove the existing property.

Note: This method is analogous to calling the corresponding data type specific method, but can be used for more generalized logic.

void **set_color**(name: StringName, theme_type: StringName, color: Color)

Creates or changes the value of the Color property defined by `name` and `theme_type`. Use clear_color to remove the property.

void **set_constant**(name: StringName, theme_type: StringName, constant: int) 🔗

Creates or changes the value of the constant property defined by `name` and `theme_type`. Use clear_constant to remove the property.

---

void **set_font**(name: StringName, theme_type: StringName, font: Font) 🔗

Creates or changes the value of the Font property defined by `name` and `theme_type`. Use clear_font to remove the property.

---

void **set_font_size**(name: StringName, theme_type: StringName, font_size: int) 🔗

Creates or changes the value of the font size property defined by `name` and `theme_type`. Use clear_font_size to remove the property.

---

void **set_icon**(name: StringName, theme_type: StringName, texture: Texture2D) 🔗

Creates or changes the value of the icon property defined by `name` and `theme_type`. Use clear_icon to remove the property.

---

void **set_stylebox**(name: StringName, theme_type: StringName, texture: StyleBox) 🔗

Creates or changes the value of the StyleBox property defined by `name` and `theme_type`. Use clear_stylebox to remove the property.

---

void **set_theme_item**(data_type: DataType, name: StringName, theme_type: StringName, value: Variant)

Creates or changes the value of the theme property of `data_type` defined by `name` and `theme_type`. Use clear_theme_item to remove the property.

Fails if the `value` type is not accepted by `data_type`.

Note: This method is analogous to calling the corresponding data type specific method, but can be used for more generalized logic.

---

void **set_type_variation**(theme_type: StringName, base_type: StringName)

Marks `theme_type` as a variation of `base_type`.

This adds `theme_type` as a suggested option for Control.theme_type_variation on a Control that is of the `base_type` class.

Variations can also be nested, i.e. `base_type` can be another variation. If a chain of variations ends with a `base_type` matching the class of the Control, the whole chain is going to be suggested as options.

Note: Suggestions only show up if this theme resource is set as the project default theme. See ProjectSettings.gui/theme/custom.

TileMapPattern

Inherits: Resource **<** RefCounted **<** Object

Holds a pattern to be copied from or pasted into TileMaps.

Description

This resource holds a set of cells to help bulk manipulations of TileMap.

A pattern always start at the `(0,0)` coordinates and cannot have cells with negative coordinates.

Methods

int	get_cell_alternative_tile(coords: Vector2i) const
Vector2i	get_cell_atlas_coords(coords: Vector2i) const
int	get_cell_source_id(coords: Vector2i) const
Vector2i	get_size() const
Array[Vector2i]	get_used_cells() const
bool	has_cell(coords: Vector2i) const

bool	is_empty() const
void	remove_cell(coords: Vector2i, update_size: bool)
void	set_cell(coords: Vector2i, source_id: int = -1, atlas_coords: Vector2i = Vector2i(-1, -1), alternative_tile: int = -1)
void	set_size(size: Vector2i)

Method Descriptions

int **get_cell_alternative_tile**(coords: Vector2i) const 🔗

Returns the tile alternative ID of the cell at `coords`.

Vector2i **get_cell_atlas_coords**(coords: Vector2i) const 🔗

Returns the tile atlas coordinates ID of the cell at `coords`.

int **get_cell_source_id**(coords: Vector2i) const 🔗

Returns the tile source ID of the cell at `coords`.

Vector2i **get_size**() const 🔗

Returns the size, in cells, of the pattern.

Array[Vector2i] **get_used_cells**() const

Returns the list of used cell coordinates in the pattern.

---

bool **has_cell**(coords: Vector2i) const

Returns whether the pattern has a tile at the given coordinates.

---

bool **is_empty**() const

Returns whether the pattern is empty or not.

---

void **remove_cell**(coords: Vector2i, update_size: bool)

Remove the cell at the given coordinates.

---

void **set_cell**(coords: Vector2i, source_id: int = -1, atlas_coords: Vector2i = Vector2i(-1, -1), alternative_tile: int = -1)

Sets the tile identifiers for the cell at coordinates `coords`. See TileMap.set_cell.

---

void **set_size**(size: Vector2i)

Sets the size of the pattern.

TileSet

Inherits: Resource **<** RefCounted **<** Object

Tile library for tilemaps.

Description

A TileSet is a library of tiles for a TileMap. A TileSet handles a list of TileSetSource, each of them storing a set of tiles.

Tiles can either be from a TileSetAtlasSource, which renders tiles out of a texture with support for physics, navigation, etc., or from a TileSetScenesCollectionSource, which exposes scene-based tiles.

Tiles are referenced by using three IDs: their source ID, their atlas coordinates ID, and their alternative tile ID.

A TileSet can be configured so that its tiles expose more or fewer properties. To do so, the TileSet resources use property layers, which you can add or remove depending on your needs.

For example, adding a physics layer allows giving collision shapes to your tiles. Each layer has dedicated properties (physics layer and mask), so you may add several TileSet physics layers for each type of collision you need.

See the functions to add new layers for more information.

Tutorials

- Using Tilemaps

- 2D Platformer Demo [https://godotengine.org/asset-library/asset/2727]
- 2D Isometric Demo [https://godotengine.org/asset-library/asset/2718]
- 2D Hexagonal Demo [https://godotengine.org/asset-library/asset/2717]
- 2D Grid-based Navigation with AStarGrid2D Demo [https://godotengine.org/asset-library/asset/2723]
- 2D Role Playing Game (RPG) Demo [https://godotengine.org/asset-library/asset/2729]
- 2D Kinematic Character Demo [https://godotengine.org/asset-library/asset/2719]

Properties

TileLayout	tile_layout	`0`
TileOffsetAxis	tile_offset_axis	`0`
TileShape	tile_shape	`0`
Vector2i	tile_size	`Vector2i(16, 16)`
bool	uv_clipping	`false`

Methods

void	add_custom_data_layer(to_position: int = -1)
void	add_navigation_layer(to_position: int = -1)

void	add_occlusion_layer(to_position: int = -1)
int	add_pattern(pattern: TileMapPattern, index: int = -1)
void	add_physics_layer(to_position: int = -1)
int	add_source(source: TileSetSource, atlas_source_id_override: int = -1)
void	add_terrain(terrain_set: int, to_position: int = -1)
void	add_terrain_set(to_position: int = -1)
void	cleanup_invalid_tile_proxies()
void	clear_tile_proxies()
Array	get_alternative_level_tile_proxy(source_from: int, coords_from: Vector2i, alternative_from: int)
Array	get_coords_level_tile_proxy(source_from: int, coords_from: Vector2i)
int	get_custom_data_layer_by_name(layer_name: String) const

String	get_custom_data_layer_name(layer_index: int) const
Variant.Type	get_custom_data_layer_type(layer_index: int) const
int	get_custom_data_layers_count() const
bool	get_navigation_layer_layer_value(layer_index: int, layer_number: int) const
int	get_navigation_layer_layers(layer_index: int) const
int	get_navigation_layers_count() const
int	get_next_source_id() const
int	get_occlusion_layer_light_mask(layer_index: int) const
bool	get_occlusion_layer_sdf_collision(layer_index: int) const
int	get_occlusion_layers_count() const
TileMapPattern	get_pattern(index: int = -1)

int	get_patterns_count()
int	get_physics_layer_collision_layer(layer_index: int) const
int	get_physics_layer_collision_mask(layer_index: int) const
PhysicsMaterial	get_physics_layer_physics_material(layer_index: int) const
int	get_physics_layers_count() const
TileSetSource	get_source(source_id: int) const
int	get_source_count() const
int	get_source_id(index: int) const
int	get_source_level_tile_proxy(source_from: int)
Color	get_terrain_color(terrain_set: int, terrain_index: int) const
String	get_terrain_name(terrain_set: int, terrain_index: int) const

TerrainMode	get_terrain_set_mode(terrain_set: int) const
int	get_terrain_sets_count() const
int	get_terrains_count(terrain_set: int) const
bool	has_alternative_level_tile_proxy(source_from: int, coords_from: Vector2i, alternative_from: int)
bool	has_coords_level_tile_proxy(source_from: int, coords_from: Vector2i)
bool	has_source(source_id: int) const
bool	has_source_level_tile_proxy(source_from: int)
Array	map_tile_proxy(source_from: int, coords_from: Vector2i, alternative_from: int) const
void	move_custom_data_layer(layer_index: int, to_position: int)
void	move_navigation_layer(layer_index: int, to_position: int)
void	move_occlusion_layer(layer_index: int, to_position: int)

void	move_physics_layer(layer_index: int, to_position: int)
void	move_terrain(terrain_set: int, terrain_index: int, to_position: int)
void	move_terrain_set(terrain_set: int, to_position: int)
void	remove_alternative_level_tile_proxy(source_from: int, coords_from: Vector2i, alternative_from: int)
void	remove_coords_level_tile_proxy(source_from: int, coords_from: Vector2i)
void	remove_custom_data_layer(layer_index: int)
void	remove_navigation_layer(layer_index: int)
void	remove_occlusion_layer(layer_index: int)
void	remove_pattern(index: int)
void	remove_physics_layer(layer_index: int)
void	remove_source(source_id: int)

void	remove_source_level_tile_proxy(source_from: int)
void	remove_terrain(terrain_set: int, terrain_index: int)
void	remove_terrain_set(terrain_set: int)
void	set_alternative_level_tile_proxy(source_from: int, coords_from: Vector2i, alternative_from: int, source_to: int, coords_to: Vector2i, alternative_to: int)
void	set_coords_level_tile_proxy(p_source_from: int, coords_from: Vector2i, source_to: int, coords_to: Vector2i)
void	set_custom_data_layer_name(layer_index: int, layer_name: String)
void	set_custom_data_layer_type(layer_index: int, layer_type: Variant.Type)
void	set_navigation_layer_layer_value(layer_index: int, layer_number: int, value: bool)
void	set_navigation_layer_layers(layer_index: int, layers: int)

void	set_occlusion_layer_light_mask(layer_index: int, light_mask: int)
void	set_occlusion_layer_sdf_collision(layer_index: int, sdf_collision: bool)
void	set_physics_layer_collision_layer(layer_index: int, layer: int)
void	set_physics_layer_collision_mask(layer_index: int, mask: int)
void	set_physics_layer_physics_material(layer_index: int, physics_material: PhysicsMaterial)
void	set_source_id(source_id: int, new_source_id: int)
void	set_source_level_tile_proxy(source_from: int, source_to: int)
void	set_terrain_color(terrain_set: int, terrain_index: int, color: Color)
void	set_terrain_name(terrain_set: int, terrain_index: int, name: String)

void	set_terrain_set_mode(terrain_set: int, mode: TerrainMode)

Enumerations

enum **TileShape**: 🔗

TileShape **TILE_SHAPE_SQUARE** = `0`

Rectangular tile shape.

TileShape **TILE_SHAPE_ISOMETRIC** = `1`

Diamond tile shape (for isometric look).

Note: Isometric **TileSet** works best if TileMap and all its layers have Y-sort enabled.

TileShape **TILE_SHAPE_HALF_OFFSET_SQUARE** = `2`

Rectangular tile shape with one row/column out of two offset by half a tile.

TileShape **TILE_SHAPE_HEXAGON** = `3`

Hexagonal tile shape.

enum **TileLayout**: 🔗

TileLayout **TILE_LAYOUT_STACKED** = `0`

Tile coordinates layout where both axis stay consistent with their respective local horizontal and vertical axis.

TileLayout **TILE_LAYOUT_STACKED_OFFSET** = `1`

Same as TILE_LAYOUT_STACKED, but the first half-offset is negative instead of positive.

TileLayout **TILE_LAYOUT_STAIRS_RIGHT** = `2`

Tile coordinates layout where the horizontal axis stay horizontal, and the vertical one goes down-right.

TileLayout **TILE_LAYOUT_STAIRS_DOWN** = `3`

Tile coordinates layout where the vertical axis stay vertical, and the horizontal one goes down-right.

TileLayout **TILE_LAYOUT_DIAMOND_RIGHT** = `4`

Tile coordinates layout where the horizontal axis goes up-right, and the vertical one goes down-right.

TileLayout **TILE_LAYOUT_DIAMOND_DOWN** = `5`

Tile coordinates layout where the horizontal axis goes down-right, and the vertical one goes down-left.

---

enum **TileOffsetAxis**: 🔗

TileOffsetAxis **TILE_OFFSET_AXIS_HORIZONTAL** = `0`

Horizontal half-offset.

TileOffsetAxis **TILE_OFFSET_AXIS_VERTICAL** = `1`

Vertical half-offset.

---

enum **CellNeighbor**: 🔗

CellNeighbor **CELL_NEIGHBOR_RIGHT_SIDE** = `0`

Neighbor on the right side.

CellNeighbor **CELL_NEIGHBOR_RIGHT_CORNER** = `1`

Neighbor in the right corner.

CellNeighbor **CELL_NEIGHBOR_BOTTOM_RIGHT_SIDE** = `2`

Neighbor on the bottom right side.

CellNeighbor **CELL_NEIGHBOR_BOTTOM_RIGHT_CORNER** = `3`

Neighbor in the bottom right corner.

CellNeighbor **CELL_NEIGHBOR_BOTTOM_SIDE** = `4`

Neighbor on the bottom side.

CellNeighbor **CELL_NEIGHBOR_BOTTOM_CORNER** = `5`

Neighbor in the bottom corner.

CellNeighbor **CELL_NEIGHBOR_BOTTOM_LEFT_SIDE** = `6`

Neighbor on the bottom left side.

CellNeighbor **CELL_NEIGHBOR_BOTTOM_LEFT_CORNER** = `7`

Neighbor in the bottom left corner.

CellNeighbor **CELL_NEIGHBOR_LEFT_SIDE** = `8`

Neighbor on the left side.

CellNeighbor **CELL_NEIGHBOR_LEFT_CORNER** = `9`

Neighbor in the left corner.

CellNeighbor **CELL_NEIGHBOR_TOP_LEFT_SIDE** = `10`

Neighbor on the top left side.

CellNeighbor **CELL_NEIGHBOR_TOP_LEFT_CORNER** = `11`

Neighbor in the top left corner.

CellNeighbor **CELL_NEIGHBOR_TOP_SIDE** = `12`

Neighbor on the top side.

CellNeighbor **CELL_NEIGHBOR_TOP_CORNER** = `13`

Neighbor in the top corner.

CellNeighbor **CELL_NEIGHBOR_TOP_RIGHT_SIDE** = `14`

Neighbor on the top right side.

CellNeighbor **CELL_NEIGHBOR_TOP_RIGHT_CORNER** = `15`

Neighbor in the top right corner.

---

enum **TerrainMode**: 🔗

TerrainMode **TERRAIN_MODE_MATCH_CORNERS_AND_SIDES** = `0`

Requires both corners and side to match with neighboring tiles' terrains.

TerrainMode **TERRAIN_MODE_MATCH_CORNERS** = `1`

Requires corners to match with neighboring tiles' terrains.

TerrainMode **TERRAIN_MODE_MATCH_SIDES** = `2`

Requires sides to match with neighboring tiles' terrains.

Property Descriptions

TileLayout **tile_layout** = `0`

- void **set_tile_layout**(value: TileLayout)
- TileLayout **get_tile_layout**()

For all half-offset shapes (Isometric, Hexagonal and Half-Offset square), changes the way tiles are indexed in the TileMap grid.

TileOffsetAxis **tile_offset_axis** = `0`

- void **set_tile_offset_axis**(value: TileOffsetAxis)
- TileOffsetAxis **get_tile_offset_axis**()

For all half-offset shapes (Isometric, Hexagonal and Half-Offset square), determines the offset axis.

TileShape **tile_shape** = `0`

- void **set_tile_shape**(value: TileShape)
- TileShape **get_tile_shape**()

The tile shape.

Vector2i **tile_size** = `Vector2i(16, 16)`

- void **set_tile_size**(value: Vector2i)
- Vector2i **get_tile_size**()

The tile size, in pixels. For all tile shapes, this size corresponds to the encompassing rectangle of the tile shape. This is thus the minimal cell size required in an atlas.

---

bool **uv_clipping** = `false`

- void **set_uv_clipping**(value: bool)
- bool **is_uv_clipping**()

Enables/Disable uv clipping when rendering the tiles.

Method Descriptions

void **add_custom_data_layer**(to_position: int = -1)

Adds a custom data layer to the TileSet at the given position `to_position` in the array. If `to_position` is -1, adds it at the end of the array.

Custom data layers allow assigning custom properties to atlas tiles.

---

void **add_navigation_layer**(to_position: int = -1)

Adds a navigation layer to the TileSet at the given position `to_position` in the array. If `to_position` is -1, adds it at the end of the array.

Navigation layers allow assigning a navigable area to atlas tiles.

---

void **add_occlusion_layer**(to_position: int = -1) 🔗

Adds an occlusion layer to the TileSet at the given position `to_position` in the array. If `to_position` is -1, adds it at the end of the array.

Occlusion layers allow assigning occlusion polygons to atlas tiles.

---

int **add_pattern**(pattern: TileMapPattern, index: int = -1) 🔗

Adds a TileMapPattern to be stored in the TileSet resource. If provided, insert it at the given `index`.

---

void **add_physics_layer**(to_position: int = -1) 🔗

Adds a physics layer to the TileSet at the given position `to_position` in the array. If `to_position` is -1, adds it at the end of the array.

Physics layers allow assigning collision polygons to atlas tiles.

---

int **add_source**(source: TileSetSource, atlas_source_id_override: int = -1) 🔗

Adds a TileSetSource to the TileSet. If `atlas_source_id_override` is not -1, also set its source ID. Otherwise, a unique identifier is automatically generated.

The function returns the added source ID or -1 if the source could not be added.

Warning: A source cannot belong to two TileSets at the same time. If the added source was attached to another **TileSet**, it will be removed from that one.

---

void **add_terrain**(terrain_set: int, to_position: int = -1)

Adds a new terrain to the given terrain set `terrain_set` at the given position `to_position` in the array. If `to_position` is -1, adds it at the end of the array.

---

void **add_terrain_set**(to_position: int = -1)

Adds a new terrain set at the given position `to_position` in the array. If `to_position` is -1, adds it at the end of the array.

---

void **cleanup_invalid_tile_proxies**()

Clears tile proxies pointing to invalid tiles.

---

void **clear_tile_proxies**()

Clears all tile proxies.

Array **get_alternative_level_tile_proxy**(source_from: int, coords_from: Vector2i, alternative_from: int)

Returns the alternative-level proxy for the given identifiers. The returned array contains the three proxie's target identifiers (source ID, atlas coords ID and alternative tile ID).

If the TileSet has no proxy for the given identifiers, returns an empty Array.

Array **get_coords_level_tile_proxy**(source_from: int, coords_from: Vector2i)

Returns the coordinate-level proxy for the given identifiers. The returned array contains the two target identifiers of the proxy (source ID and atlas coordinates ID).

If the TileSet has no proxy for the given identifiers, returns an empty Array.

---

int **get_custom_data_layer_by_name**(layer_name: String) const 🔗

Returns the index of the custom data layer identified by the given name.

String **get_custom_data_layer_name**(layer_index: int) const 🔗

Returns the name of the custom data layer identified by the given index.

Variant.Type **get_custom_data_layer_type**(layer_index: int) const 🔗

Returns the type of the custom data layer identified by the given index.

---

int **get_custom_data_layers_count**() const 🔗

Returns the custom data layers count.

---

bool **get_navigation_layer_layer_value**(layer_index: int, layer_number: int) const 🔗

Returns whether or not the specified navigation layer of the TileSet navigation data layer identified by the given `layer_index` is enabled, given a navigation_layers `layer_number` between 1 and 32.

---

int **get_navigation_layer_layers**(layer_index: int) const

Returns the navigation layers (as in the Navigation server) of the given TileSet navigation layer.

---

int **get_navigation_layers_count**() const

Returns the navigation layers count.

---

int **get_next_source_id**() const

Returns a new unused source ID. This generated ID is the same that a call to add_source would return.

---

int **get_occlusion_layer_light_mask**(layer_index: int) const

Returns the light mask of the occlusion layer.

---

bool **get_occlusion_layer_sdf_collision**(layer_index: int) const

Returns if the occluders from this layer use `sdf_collision`.

---

int **get_occlusion_layers_count**() const

Returns the occlusion layers count.

TileMapPattern **get_pattern**(index: int = -1)

Returns the TileMapPattern at the given `index`.

---

int **get_patterns_count**()

Returns the number of TileMapPattern this tile set handles.

---

int **get_physics_layer_collision_layer**(layer_index: int) const

Returns the collision layer (as in the physics server) bodies on the given TileSet's physics layer are in.

---

int **get_physics_layer_collision_mask**(layer_index: int) const

Returns the collision mask of bodies on the given TileSet's physics layer.

PhysicsMaterial **get_physics_layer_physics_material**(layer_index: int) const

Returns the physics material of bodies on the given TileSet's physics layer.

---

int **get_physics_layers_count**() const

Returns the physics layers count.

TileSetSource **get_source**(source_id: int) const

Returns the TileSetSource with ID `source_id`.

---

int **get_source_count**() const 🔗

Returns the number of TileSetSource in this TileSet.

---

int **get_source_id**(index: int) const 🔗

Returns the source ID for source with index `index`.

---

int **get_source_level_tile_proxy**(source_from: int) 🔗

Returns the source-level proxy for the given source identifier.

If the TileSet has no proxy for the given identifier, returns -1.

Color **get_terrain_color**(terrain_set: int, terrain_index: int) const 🔗

Returns a terrain's color.

String **get_terrain_name**(terrain_set: int, terrain_index: int) const 🔗

Returns a terrain's name.

TerrainMode **get_terrain_set_mode**(terrain_set: int) const 🔗

Returns a terrain set mode.

---

int **get_terrain_sets_count**() const

Returns the terrain sets count.

int **get_terrains_count**(terrain_set: int) const

Returns the number of terrains in the given terrain set.

bool **has_alternative_level_tile_proxy**(source_from: int, coords_from: Vector2i, alternative_from: int)

Returns if there is an alternative-level proxy for the given identifiers.

bool **has_coords_level_tile_proxy**(source_from: int, coords_from: Vector2i)

Returns if there is a coodinates-level proxy for the given identifiers.

bool **has_source**(source_id: int) const

Returns if this TileSet has a source for the given source ID.

bool **has_source_level_tile_proxy**(source_from: int)

Returns if there is a source-level proxy for the given source ID.

Array **map_tile_proxy**(source_from: int, coords_from: Vector2i, alternative_from: int) const

According to the configured proxies, maps the provided identifiers to a new set of identifiers. The source ID, atlas coordinates ID and alternative tile ID are returned as a 3 elements Array.

This function first look for matching alternative-level proxies, then coordinates-level proxies, then source-level proxies.

If no proxy corresponding to provided identifiers are found, returns the same values the ones used as arguments.

---

void **move_custom_data_layer**(layer_index: int, to_position: int)

Moves the custom data layer at index `layer_index` to the given position `to_position` in the array. Also updates the atlas tiles accordingly.

---

void **move_navigation_layer**(layer_index: int, to_position: int)

Moves the navigation layer at index `layer_index` to the given position `to_position` in the array. Also updates the atlas tiles accordingly.

---

void **move_occlusion_layer**(layer_index: int, to_position: int)

Moves the occlusion layer at index `layer_index` to the given position `to_position` in the array. Also updates the atlas tiles accordingly.

---

void **move_physics_layer**(layer_index: int, to_position: int)

Moves the physics layer at index `layer_index` to the given position `to_position` in the array. Also updates the atlas tiles accordingly.

---

void **move_terrain**(terrain_set: int, terrain_index: int, to_position: int)

Moves the terrain at index `terrain_index` for terrain set `terrain_set` to the given position `to_position` in the array. Also updates the atlas tiles accordingly.

void **move_terrain_set**(terrain_set: int, to_position: int)

Moves the terrain set at index `terrain_set` to the given position `to_position` in the array. Also updates the atlas tiles accordingly.

void **remove_alternative_level_tile_proxy**(source_from: int, coords_from: Vector2i, alternative_from: int)

Removes an alternative-level proxy for the given identifiers.

void **remove_coords_level_tile_proxy**(source_from: int, coords_from: Vector2i)

Removes a coordinates-level proxy for the given identifiers.

void **remove_custom_data_layer**(layer_index: int)

Removes the custom data layer at index `layer_index`. Also updates the atlas tiles accordingly.

void **remove_navigation_layer**(layer_index: int)

Removes the navigation layer at index `layer_index`. Also updates the atlas tiles accordingly.

---

void **remove_occlusion_layer**(layer_index: int)

Removes the occlusion layer at index `layer_index`. Also updates the atlas tiles accordingly.

---

void **remove_pattern**(index: int)

Remove the TileMapPattern at the given index.

---

void **remove_physics_layer**(layer_index: int)

Removes the physics layer at index `layer_index`. Also updates the atlas tiles accordingly.

---

void **remove_source**(source_id: int)

Removes the source with the given source ID.

---

void **remove_source_level_tile_proxy**(source_from: int)

Removes a source-level tile proxy.

---

void **remove_terrain**(terrain_set: int, terrain_index: int)

Removes the terrain at index `terrain_index` in the given terrain set `terrain_set`. Also updates the atlas tiles accordingly.

---

void **remove_terrain_set**(terrain_set: int)

Removes the terrain set at index `terrain_set`. Also updates the atlas tiles accordingly.

void **set_alternative_level_tile_proxy**(source_from: int, coords_from: Vector2i, alternative_from: int, source_to: int, coords_to: Vector2i, alternative_to: int)

Create an alternative-level proxy for the given identifiers. A proxy will map set of tile identifiers to another set of identifiers.

This can be used to replace a tile in all TileMaps using this TileSet, as TileMap nodes will find and use the proxy's target tile when one is available.

Proxied tiles can be automatically replaced in TileMap nodes using the editor.

void **set_coords_level_tile_proxy**(p_source_from: int, coords_from: Vector2i, source_to: int, coords_to: Vector2i)

Creates a coordinates-level proxy for the given identifiers. A proxy will map set of tile identifiers to another set of identifiers. The alternative tile ID is kept the same when using coordinates-level proxies.

This can be used to replace a tile in all TileMaps using this TileSet, as TileMap nodes will find and use the proxy's target tile when one is available.

Proxied tiles can be automatically replaced in TileMap nodes using the editor.

void **set_custom_data_layer_name**(layer_index: int, layer_name: String)

Sets the name of the custom data layer identified by the given index. Names are identifiers of the layer therefore if the name is already taken it will fail and raise an error.

---

void **set_custom_data_layer_type**(layer_index: int, layer_type: Variant.Type)

Sets the type of the custom data layer identified by the given index.

---

void **set_navigation_layer_layer_value**(layer_index: int, layer_number: int, value: bool)

Based on `value`, enables or disables the specified navigation layer of the TileSet navigation data layer identified by the given `layer_index`, given a navigation_layers `layer_number` between 1 and 32.

---

void **set_navigation_layer_layers**(layer_index: int, layers: int)

Sets the navigation layers (as in the navigation server) for navigation regions in the given TileSet navigation layer.

---

void **set_occlusion_layer_light_mask**(layer_index: int, light_mask: int)

Sets the occlusion layer (as in the rendering server) for occluders in the given TileSet occlusion layer.

---

void **set_occlusion_layer_sdf_collision**(layer_index: int, sdf_collision: bool)

Enables or disables SDF collision for occluders in the given TileSet occlusion layer.

void **set_physics_layer_collision_layer**(layer_index: int, layer: int)

Sets the physics layer (as in the physics server) for bodies in the given TileSet physics layer.

void **set_physics_layer_collision_mask**(layer_index: int, mask: int)

Sets the physics layer (as in the physics server) for bodies in the given TileSet physics layer.

void **set_physics_layer_physics_material**(layer_index: int, physics_material: PhysicsMaterial)

Sets the physics material for bodies in the given TileSet physics layer.

void **set_source_id**(source_id: int, new_source_id: int)

Changes a source's ID.

void **set_source_level_tile_proxy**(source_from: int, source_to: int)

Creates a source-level proxy for the given source ID. A proxy will map set of tile identifiers to another set of identifiers. Both the atlas coordinates ID and the alternative tile ID are kept the same when using source-level proxies.

This can be used to replace a source in all TileMaps using this TileSet, as TileMap nodes will find and use the proxy's target source when one is available.

Proxied tiles can be automatically replaced in TileMap nodes using the editor.

---

void **set_terrain_color**(terrain_set: int, terrain_index: int, color: Color)

Sets a terrain's color. This color is used for identifying the different terrains in the TileSet editor.

---

void **set_terrain_name**(terrain_set: int, terrain_index: int, name: String)

Sets a terrain's name.

---

void **set_terrain_set_mode**(terrain_set: int, mode: TerrainMode)

Sets a terrain mode. Each mode determines which bits of a tile shape is used to match the neighboring tiles' terrains.

TileSetAtlasSource

Inherits: TileSetSource **<** Resource **<** RefCounted **<** Object

Exposes a 2D atlas texture as a set of tiles for a TileSet resource.

Description

An atlas is a grid of tiles laid out on a texture. Each tile in the grid must be exposed using create_tile. Those tiles are then indexed using their coordinates in the grid.

Each tile can also have a size in the grid coordinates, making it more or less cells in the atlas.

Alternatives version of a tile can be created using create_alternative_tile, which are then indexed using an alternative ID. The main tile (the one in the grid), is accessed with an alternative ID equal to 0.

Each tile alternate has a set of properties that is defined by the source's TileSet layers. Those properties are stored in a TileData object that can be accessed and modified using get_tile_data.

As TileData properties are stored directly in the TileSetAtlasSource resource, their properties might also be set using `TileSetAtlasSource.set("<coords_x>:<coords_y>/<alternative_id>/<tile_data_property>")`.

Properties

Vector2i	margins	`Vector2i(0, 0)`

Vector2i	separation	`Vector2i(0, 0)`
Texture2D	texture	
Vector2i	texture_region_size	`Vector2i(16, 16)`
bool	use_texture_padding	`true`

Methods

void	clear_tiles_outside_texture()
int	create_alternative_tile(atlas_coords: Vector2i, alternative_id_override: int = -1)
void	create_tile(atlas_coords: Vector2i, size: Vector2i = Vector2i(1, 1))
Vector2i	get_atlas_grid_size() const
int	get_next_alternative_tile_id(atlas_coords: Vector2i) const
Texture2D	get_runtime_texture() const

Rect2i	get_runtime_tile_texture_region(atlas_coords: Vector2i, frame: int) const
int	get_tile_animation_columns(atlas_coords: Vector2i) const
float	get_tile_animation_frame_duration(atlas_coords: Vector2i, frame_index: int) const
int	get_tile_animation_frames_count(atlas_coords: Vector2i) const
TileAnimationMode	get_tile_animation_mode(atlas_coords: Vector2i) const
Vector2i	get_tile_animation_separation(atlas_coords: Vector2i) const
float	get_tile_animation_speed(atlas_coords: Vector2i) const
float	get_tile_animation_total_duration(atlas_coords: Vector2i) const
Vector2i	get_tile_at_coords(atlas_coords: Vector2i) const
TileData	get_tile_data(atlas_coords: Vector2i, alternative_tile: int) const

Vector2i	get_tile_size_in_atlas(atlas_coords: Vector2i) const
Rect2i	get_tile_texture_region(atlas_coords: Vector2i, frame: int = 0) const
PackedVector2Array	get_tiles_to_be_removed_on_change(texture: Texture2D, margins: Vector2i, separation: Vector2i, texture_region_size: Vector2i)
bool	has_room_for_tile(atlas_coords: Vector2i, size: Vector2i, animation_columns: int, animation_separation: Vector2i, frames_count: int, ignored_tile: Vector2i = Vector2i(-1, -1)) const
bool	has_tiles_outside_texture() const
void	move_tile_in_atlas(atlas_coords: Vector2i, new_atlas_coords: Vector2i = Vector2i(-1, -1), new_size: Vector2i = Vector2i(-1, -1))
void	remove_alternative_tile(atlas_coords: Vector2i, alternative_tile: int)
void	remove_tile(atlas_coords: Vector2i)
void	set_alternative_tile_id(atlas_coords: Vector2i, alternative_tile: int, new_id: int)

void	set_tile_animation_columns(atlas_coords: Vector2i, frame_columns: int)
void	set_tile_animation_frame_duration(atlas_coords: Vector2i, frame_index: int, duration: float)
void	set_tile_animation_frames_count(atlas_coords: Vector2i, frames_count: int)
void	set_tile_animation_mode(atlas_coords: Vector2i, mode: TileAnimationMode)
void	set_tile_animation_separation(atlas_coords: Vector2i, separation: Vector2i)
void	set_tile_animation_speed(atlas_coords: Vector2i, speed: float)

Enumerations

enum **TileAnimationMode**: 🔗

TileAnimationMode **TILE_ANIMATION_MODE_DEFAULT** = `0`

Tile animations start at same time, looking identical.

TileAnimationMode **TILE_ANIMATION_MODE_RANDOM_START_TIMES** = `1`

Tile animations start at random times, looking varied.

TileAnimationMode **TILE_ANIMATION_MODE_MAX** = `2`

Represents the size of the TileAnimationMode enum.

Constants

TRANSFORM_FLIP_H = `4096`

Represents cell's horizontal flip flag. Should be used directly with TileMap to flip placed tiles by altering their alternative IDs.

```
var alternate_id = $TileMap.get_cell_alternative_tile(0, Vector2i(2, 2))
if not alternate_id & TileSetAtlasSource.TRANSFORM_FLIP_H:
    # If tile is not already flipped, flip it.
    $TileMap.set_cell(0, Vector2i(2, 2), source_id, atlas_coords, alternate_id | TileSetAtlasSource.TRANSFORM_FLIP_H)
```

Note: These transformations can be combined to do the equivalent of 0, 90, 180, and 270 degree rotations, as shown below:

```
enum TileTransform {
    ROTATE_0 = 0,
    ROTATE_90 = TileSetAtlasSource.TRANSFORM_TRANSPOSE | TileSetAtlasSource.TRANSFORM_FLIP_H,
    ROTATE_180 = TileSetAtlasSource.TRANSFORM_FLIP_H | TileSetAtlasSource.TRANSFORM_FLIP_V,
    ROTATE_270 = TileSetAtlasSource.TRANSFORM_TRANSPOSE | TileSetAtlasSource.TRANSFORM_FLIP_V,
}
```

TRANSFORM_FLIP_V = `8192`

Represents cell's vertical flip flag. See TRANSFORM_FLIP_H for usage.

TRANSFORM_TRANSPOSE = `16384`

Represents cell's transposed flag. See TRANSFORM_FLIP_H for usage.

Property Descriptions

Vector2i **margins** = `Vector2i(0, 0)`

- void **set_margins**(value: Vector2i)
- Vector2i **get_margins**()

Margins, in pixels, to offset the origin of the grid in the texture.

Vector2i **separation** = `Vector2i(0, 0)`

- void **set_separation**(value: Vector2i)
- Vector2i **get_separation**()

Separation, in pixels, between each tile texture region of the grid.

Texture2D **texture**

- void **set_texture**(value: Texture2D)
- Texture2D **get_texture**()

The atlas texture.

Vector2i **texture_region_size** = `Vector2i(16, 16)`

- void **set_texture_region_size**(value: Vector2i)
- Vector2i **get_texture_region_size**()

The base tile size in the texture (in pixel). This size must be bigger than the TileSet's `tile_size` value.

---

bool **use_texture_padding** = `true`

- void **set_use_texture_padding**(value: bool)
- bool **get_use_texture_padding**()

If `true`, generates an internal texture with an additional one pixel padding around each tile. Texture padding avoids a common artifact where lines appear between tiles.

Disabling this setting might lead a small performance improvement, as generating the internal texture requires both memory and processing time when the TileSetAtlasSource resource is modified.

Method Descriptions

void **clear_tiles_outside_texture**()

Removes all tiles that don't fit the available texture area. This method iterates over all the source's tiles, so it's advised to use has_tiles_outside_texture beforehand.

int **create_alternative_tile**(atlas_coords: Vector2i, alternative_id_override: int = -1)

Creates an alternative tile for the tile at coordinates `atlas_coords`. If `alternative_id_override` is -1, give it an automatically generated unique ID, or assigns it the given ID otherwise.

Returns the new alternative identifier, or -1 if the alternative could not be created with a provided `alternative_id_override`.

void **create_tile**(atlas_coords: Vector2i, size: Vector2i = Vector2i(1, 1))

Creates a new tile at coordinates `atlas_coords` with the given `size`.

Vector2i **get_atlas_grid_size**() const

Returns the atlas grid size, which depends on how many tiles can fit in the texture. It thus depends on the texture's size, the atlas margins, and the tiles' texture_region_size.

int **get_next_alternative_tile_id**(atlas_coords: Vector2i) const

Returns the alternative ID a following call to create_alternative_tile would return.

Texture2D **get_runtime_texture**() const

If use_texture_padding is `false`, returns texture. Otherwise, returns and internal ImageTexture created that includes the padding.

Rect2i **get_runtime_tile_texture_region**(atlas_coords: Vector2i, frame: int) const

Returns the region of the tile at coordinates `atlas_coords` for the given `frame` inside the texture returned by get_runtime_texture.

Note: If use_texture_padding is `false`, returns the same as get_tile_texture_region.

int **get_tile_animation_columns**(atlas_coords: Vector2i) const

Returns how many columns the tile at `atlas_coords` has in its animation layout.

float **get_tile_animation_frame_duration**(atlas_coords: Vector2i, frame_index: int) const

Returns the animation frame duration of frame `frame_index` for the tile at coordinates `atlas_coords`.

---

int **get_tile_animation_frames_count**(atlas_coords: Vector2i) const

Returns how many animation frames has the tile at coordinates `atlas_coords`.

TileAnimationMode **get_tile_animation_mode**(atlas_coords: Vector2i) const

Returns the tile animation mode of the tile at `atlas_coords`. See also set_tile_animation_mode.

Vector2i **get_tile_animation_separation**(atlas_coords: Vector2i) const

Returns the separation (as in the atlas grid) between each frame of an animated tile at coordinates `atlas_coords`.

---

float **get_tile_animation_speed**(atlas_coords: Vector2i) const

Returns the animation speed of the tile at coordinates `atlas_coords`.

---

float **get_tile_animation_total_duration**(atlas_coords: Vector2i) const

Returns the sum of the sum of the frame durations of the tile at coordinates `atlas_coords`. This value needs to be divided by the animation speed to get the actual animation loop duration.

Vector2i **get_tile_at_coords**(atlas_coords: Vector2i) const

If there is a tile covering the `atlas_coords` coordinates, returns the top-left coordinates of the tile (thus its coordinate ID). Returns `Vector2i(-1, -1)` otherwise.

TileData **get_tile_data**(atlas_coords: Vector2i, alternative_tile: int) const

Returns the TileData object for the given atlas coordinates and alternative ID.

Vector2i **get_tile_size_in_atlas**(atlas_coords: Vector2i) const

Returns the size of the tile (in the grid coordinates system) at coordinates `atlas_coords`.

Rect2i **get_tile_texture_region**(atlas_coords: Vector2i, frame: int = 0) const

Returns a tile's texture region in the atlas texture. For animated tiles, a `frame` argument might be provided for the different frames of the animation.

PackedVector2Array **get_tiles_to_be_removed_on_change**(texture: Texture2D, margins: Vector2i, separation: Vector2i, texture_region_size: Vector2i)

Returns an array of tiles coordinates ID that will be automatically removed when modifying one or several of those properties: `texture`, `margins`, `separation` or `texture_region_size`. This can be used to undo changes that would have caused tiles data loss.

bool **has_room_for_tile**(atlas_coords: Vector2i, size: Vector2i, animation_columns: int, animation_separation: Vector2i, frames_count: int, ignored_tile: Vector2i = Vector2i(-1, -1)) const

Returns whether there is enough room in an atlas to create/modify a tile with the given properties. If `ignored_tile` is provided, act as is the given tile was not present in the atlas. This may be used when you want to modify a tile's properties.

bool **has_tiles_outside_texture**() const

Checks if the source has any tiles that don't fit the texture area (either partially or completely).

void **move_tile_in_atlas**(atlas_coords: Vector2i, new_atlas_coords: Vector2i = Vector2i(-1, -1), new_size: Vector2i = Vector2i(-1, -1))

Move the tile and its alternatives at the `atlas_coords` coordinates to the `new_atlas_coords` coordinates with the `new_size` size. This functions will fail if a tile is already present in the given area.

If `new_atlas_coords` is `Vector2i(-1, -1)`, keeps the tile's coordinates. If `new_size` is `Vector2i(-1, -1)`, keeps the tile's size.

To avoid an error, first check if a move is possible using has_room_for_tile.

void **remove_alternative_tile**(atlas_coords: Vector2i, alternative_tile: int)

Remove a tile's alternative with alternative ID `alternative_tile`.

Calling this function with `alternative_tile` equals to 0 will fail, as the base tile alternative cannot be removed.

---

void **remove_tile**(atlas_coords: Vector2i)

Remove a tile and its alternative at coordinates `atlas_coords`.

---

void **set_alternative_tile_id**(atlas_coords: Vector2i, alternative_tile: int, new_id: int)

Change a tile's alternative ID from `alternative_tile` to `new_id`.

Calling this function with `new_id` of 0 will fail, as the base tile alternative cannot be moved.

---

void **set_tile_animation_columns**(atlas_coords: Vector2i, frame_columns: int)

Sets the number of columns in the animation layout of the tile at coordinates `atlas_coords`. If set to 0, then the different frames of the animation are laid out as a single horizontal line in the atlas.

---

void **set_tile_animation_frame_duration**(atlas_coords: Vector2i, frame_index: int, duration: float)

Sets the animation frame `duration` of frame `frame_index` for the tile at coordinates `atlas_coords`.

---

void **set_tile_animation_frames_count**(atlas_coords: Vector2i, frames_count: int)

Sets how many animation frames the tile at coordinates `atlas_coords` has.

void **set_tile_animation_mode**(atlas_coords: Vector2i, mode: TileAnimationMode)

Sets the tile animation mode of the tile at `atlas_coords` to `mode`. See also get_tile_animation_mode.

void **set_tile_animation_separation**(atlas_coords: Vector2i, separation: Vector2i)

Sets the margin (in grid tiles) between each tile in the animation layout of the tile at coordinates `atlas_coords` has.

void **set_tile_animation_speed**(atlas_coords: Vector2i, speed: float)

Sets the animation speed of the tile at coordinates `atlas_coords` has.

TileSetScenesCollectionSource

Inherits: TileSetSource < Resource < RefCounted < Object

Exposes a set of scenes as tiles for a TileSet resource.

Description

When placed on a TileMap, tiles from **TileSetScenesCollectionSource** will automatically instantiate an associated scene at the cell's position in the TileMap.

Scenes are instantiated as children of the TileMap when it enters the tree. If you add/remove a scene tile in the TileMap that is already inside the tree, the TileMap will automatically instantiate/free the scene accordingly.

Note: Scene tiles all occupy one tile slot and instead use alternate tile ID to identify scene index. TileSetSource.get_tiles_count will always return `1`. Use get_scene_tiles_count to get a number of scenes in a **TileSetScenesCollectionSource**.

Use this code if you want to find the scene path at a given tile in TileMapLayer:

GDScript

```
var source_id = tile_map_layer.get_cell_source_id(Vector2i(x,
y))
if source_id > -1:
    var scene_source =
tile_map_layer.tile_set.get_source(source_id)
```

```
    if scene_source is TileSetScenesCollectionSource:
        var alt_id =
tile_map_layer.get_cell_alternative_tile(Vector2i(x, y))
        # The assigned PackedScene.
        var scene = scene_source.get_scene_tile_scene(alt_id)
```

C#

```
int sourceId = tileMapLayer.GetCellSourceId(new Vector2I(x,
y));
if (sourceId > -1)
{
    TileSetSource source =
tileMapLayer.TileSet.GetSource(sourceId);
    if (source is TileSetScenesCollectionSource sceneSource)
    {
        int altId = tileMapLayer.GetCellAlternativeTile(new
Vector2I(x, y));
        // The assigned PackedScene.
        PackedScene scene =
sceneSource.GetSceneTileScene(altId);
    }
}
```

Methods

int	create_scene_tile(packed_scene: PackedScene, id_override: int = -1)
int	get_next_scene_tile_id() const
bool	get_scene_tile_display_placeholder(id: int) const
int	get_scene_tile_id(index: int)

PackedScene	get_scene_tile_scene(id: int) const
int	get_scene_tiles_count()
bool	has_scene_tile_id(id: int)
void	remove_scene_tile(id: int)
void	set_scene_tile_display_placeholder(id: int, display_placeholder: bool)
void	set_scene_tile_id(id: int, new_id: int)
void	set_scene_tile_scene(id: int, packed_scene: PackedScene)

Method Descriptions

int **create_scene_tile**(packed_scene: PackedScene, id_override: int = -1) 🔗

Creates a scene-based tile out of the given scene.

Returns a newly generated unique ID.

---

int **get_next_scene_tile_id**() const 🔗

Returns the scene ID a following call to create_scene_tile would return.

---

bool **get_scene_tile_display_placeholder**(id: int) const

Returns whether the scene tile with `id` displays a placeholder in the editor.

---

int **get_scene_tile_id**(index: int)

Returns the scene tile ID of the scene tile at `index`.

PackedScene **get_scene_tile_scene**(id: int) const

Returns the PackedScene resource of scene tile with `id`.

---

int **get_scene_tiles_count**()

Returns the number or scene tiles this TileSet source has.

---

bool **has_scene_tile_id**(id: int)

Returns whether this TileSet source has a scene tile with `id`.

---

void **remove_scene_tile**(id: int)

Remove the scene tile with `id`.

---

void **set_scene_tile_display_placeholder**(id: int, display_placeholder: bool)

Sets whether or not the scene tile with `id` should display a placeholder in the editor. This might be useful for scenes that are not visible.

---

void **set_scene_tile_id**(id: int, new_id: int)

Changes a scene tile's ID from `id` to `new_id`. This will fail if there is already a tile with an ID equal to `new_id`.

---

void **set_scene_tile_scene**(id: int, packed_scene: PackedScene)

Assigns a PackedScene resource to the scene tile with `id`. This will fail if the scene does not extend CanvasItem, as positioning properties are needed to place the scene on the TileMap.

TileSetSource

Inherits: Resource **<** RefCounted **<** Object

Inherited By: TileSetAtlasSource, TileSetScenesCollectionSource

Exposes a set of tiles for a TileSet resource.

Description

Exposes a set of tiles for a TileSet resource.

Tiles in a source are indexed with two IDs, coordinates ID (of type Vector2i) and an alternative ID (of type int), named according to their use in the TileSetAtlasSource class.

Depending on the TileSet source type, those IDs might have restrictions on their values, this is why the base **TileSetSource** class only exposes getters for them.

You can iterate over all tiles exposed by a TileSetSource by first iterating over coordinates IDs using get_tiles_count and get_tile_id, then over alternative IDs using get_alternative_tiles_count and get_alternative_tile_id.

Warning: TileSetSource can only be added to one TileSet at the same time. Calling TileSet.add_source on a second TileSet will remove the source from the first one.

Methods

int	get_alternative_tile_id(atlas_coords: Vector2i, index: int) const
int	get_alternative_tiles_count(atlas_coords: Vector2i) const
Vector2i	get_tile_id(index: int) const
int	get_tiles_count() const
bool	has_alternative_tile(atlas_coords: Vector2i, alternative_tile: int) const
bool	has_tile(atlas_coords: Vector2i) const

Method Descriptions

int **get_alternative_tile_id**(atlas_coords: Vector2i, index: int) const 🔗

Returns the alternative ID for the tile with coordinates ID `atlas_coords` at index `index`.

---

int **get_alternative_tiles_count**(atlas_coords: Vector2i) const 🔗

Returns the number of alternatives tiles for the coordinates ID `atlas_coords`.

For TileSetAtlasSource, this always return at least 1, as the base tile with ID 0 is always part of the alternatives list.

Returns -1 if there is not tile at the given coords.

Vector2i **get_tile_id**(index: int) const

Returns the tile coordinates ID of the tile with index `index`.

---

int **get_tiles_count**() const

Returns how many tiles this atlas source defines (not including alternative tiles).

---

bool **has_alternative_tile**(atlas_coords: Vector2i, alternative_tile: int) const

Returns if the base tile at coordinates `atlas_coords` has an alternative with ID `alternative_tile`.

---

bool **has_tile**(atlas_coords: Vector2i) const

Returns if this atlas has a tile with coordinates ID `atlas_coords`.

TorusMesh

Inherits: PrimitiveMesh **<** Mesh **<** Resource **<** RefCounted **<** Object

Class representing a torus PrimitiveMesh.

Description

Class representing a torus PrimitiveMesh.

Properties

float	inner_radius	`0.5`
float	outer_radius	`1.0`
int	ring_segments	`32`
int	rings	`64`

Property Descriptions

float **inner_radius** = `0.5` 🔗

- void **set_inner_radius**(value: float)

- float **get_inner_radius**()

The inner radius of the torus.

float **outer_radius** = `1.0`

- void **set_outer_radius**(value: float)
- float **get_outer_radius**()

The outer radius of the torus.

int **ring_segments** = `32`

- void **set_ring_segments**(value: int)
- int **get_ring_segments**()

The number of edges each ring of the torus is constructed of.

int **rings** = `64`

- void **set_rings**(value: int)
- int **get_rings**()

The number of slices the torus is constructed of.

Translation

Inherits: Resource **<** RefCounted **<** Object

Inherited By: OptimizedTranslation

A language translation that maps a collection of strings to their individual translations.

Description

Translations are resources that can be loaded and unloaded on demand. They map a collection of strings to their individual translations, and they also provide convenience methods for pluralization.

Tutorials

- Internationalizing games
- Locales

Properties

String	locale	`"en"`

Methods

StringName	_get_message(src_message: StringName, context: StringName) virtual const
StringName	_get_plural_message(src_message: StringName, src_plural_message: StringName, n: int, context: StringName) virtual const
void	add_message(src_message: StringName, xlated_message: StringName, context: StringName = &"")
void	add_plural_message(src_message: StringName, xlated_messages: PackedStringArray, context: StringName = &"")
void	erase_message(src_message: StringName, context: StringName = &"")
StringName	get_message(src_message: StringName, context: StringName = &"") const
int	get_message_count() const
PackedStringArray	get_message_list() const
StringName	get_plural_message(src_message: StringName, src_plural_message: StringName, n: int, context: StringName = &"") const

---

PackedStringArray get_translated_message_list() const

Property Descriptions

String **locale** = `"en"`

- void **set_locale**(value: String)
- String **get_locale**()

The locale of the translation.

Method Descriptions

StringName **_get_message**(src_message: StringName, context: StringName) virtual const

Virtual method to override get_message.

StringName **_get_plural_message**(src_message: StringName, src_plural_message: StringName, n: int, context: StringName) virtual const

Virtual method to override get_plural_message.

---

void **add_message**(src_message: StringName, xlated_message: StringName, context: StringName = &"")

Adds a message if nonexistent, followed by its translation.

An additional context could be used to specify the translation context or differentiate polysemic words.

void **add_plural_message**(src_message: StringName, xlated_messages: PackedStringArray, context: StringName = &"")

Adds a message involving plural translation if nonexistent, followed by its translation.

An additional context could be used to specify the translation context or differentiate polysemic words.

void **erase_message**(src_message: StringName, context: StringName = &"")

Erases a message.

StringName **get_message**(src_message: StringName, context: StringName = &"") const

Returns a message's translation.

int **get_message_count**() const

Returns the number of existing messages.

PackedStringArray **get_message_list**() const

Returns all the messages (keys).

StringName **get_plural_message**(src_message: StringName, src_plural_message: StringName, n: int, context: StringName = &"") const

Returns a message's translation involving plurals.

The number `n` is the number or quantity of the plural object. It will be used to guide the translation system to fetch the correct plural form for the selected language.

PackedStringArray **get_translated_message_list**() const

Returns all the messages (translated text).

TubeTrailMesh

Inherits: PrimitiveMesh **<** Mesh **<** Resource **<** RefCounted **<** Object

Represents a straight tube-shaped PrimitiveMesh with variable width.

Description

TubeTrailMesh represents a straight tube-shaped mesh with variable width. The tube is composed of a number of cylindrical sections, each with the same section_length and number of section_rings. A curve is sampled along the total length of the tube, meaning that the curve determines the radius of the tube along its length.

This primitive mesh is usually used for particle trails.

Tutorials

- 3D Particle trails
- Particle systems (3D)

Properties

bool	cap_bottom	`true`
bool	cap_top	`true`

Curve	curve	
int	radial_steps	`8`
float	radius	`0.5`
float	section_length	`0.2`
int	section_rings	`3`
int	sections	`5`

Property Descriptions

bool **cap_bottom** = `true` 🔗

- void **set_cap_bottom**(value: bool)
- bool **is_cap_bottom**()

If `true`, generates a cap at the bottom of the tube. This can be set to `false` to speed up generation and rendering when the cap is never seen by the camera.

---

bool **cap_top** = `true` 🔗

- void **set_cap_top**(value: bool)
- bool **is_cap_top**()

If `true`, generates a cap at the top of the tube. This can be set to `false` to speed up generation and rendering when the cap is never seen by the camera.

Curve **curve**

- void **set_curve**(value: Curve)
- Curve **get_curve**()

Determines the radius of the tube along its length. The radius of a particular section ring is obtained by multiplying the baseline radius by the value of this curve at the given distance. For values smaller than `0`, the faces will be inverted.

---

int **radial_steps** = `8`

- void **set_radial_steps**(value: int)
- int **get_radial_steps**()

The number of sides on the tube. For example, a value of `5` means the tube will be pentagonal. Higher values result in a more detailed tube at the cost of performance.

---

float **radius** = `0.5`

- void **set_radius**(value: float)
- float **get_radius**()

The baseline radius of the tube. The radius of a particular section ring is obtained by multiplying this radius by the value of the curve at the given distance.

---

float **section_length** = `0.2`

- void **set_section_length**(value: float)
- float **get_section_length**()

The length of a section of the tube.

int **section_rings** = 3 🔗

- void **set_section_rings**(value: int)
- int **get_section_rings**()

The number of rings in a section. The curve is sampled on each ring to determine its radius. Higher values result in a more detailed tube at the cost of performance.

int **sections** = 5 🔗

- void **set_sections**(value: int)
- int **get_sections**()

The total number of sections on the tube.

VideoStream

Inherits: Resource **<** RefCounted **<** Object

Inherited By: VideoStreamTheora

Base resource for video streams.

Description

Base resource type for all video streams. Classes that derive from **VideoStream** can all be used as resource types to play back videos in VideoStreamPlayer.

Tutorials

- Playing videos
- Runtime file loading and saving

Properties

String	file	""

Methods

VideoStreamPlayback	_instantiate_playback() virtual

Property Descriptions

String **file** = `""` 🔗

- void **set_file**(value: String)
- String **get_file**()

The video file path or URI that this **VideoStream** resource handles.

For VideoStreamTheora, this filename should be an Ogg Theora video file with the `.ogv` extension.

Method Descriptions

VideoStreamPlayback **_instantiate_playback**() virtual 🔗

Called when the video starts playing, to initialize and return a subclass of VideoStreamPlayback.

VideoStreamPlayback

Inherits: Resource **<** RefCounted **<** Object

Internal class used by VideoStream to manage playback state when played from a VideoStreamPlayer.

Description

This class is intended to be overridden by video decoder extensions with custom implementations of VideoStream.

Methods

int	_get_channels() virtual const
float	_get_length() virtual const
int	_get_mix_rate() virtual const
float	_get_playback_position() virtual const
Texture2D	_get_texture() virtual const
bool	_is_paused() virtual const

bool	_is_playing() virtual const
void	_play() virtual
void	_seek(time: float) virtual
void	_set_audio_track(idx: int) virtual
void	_set_paused(paused: bool) virtual
void	_stop() virtual
void	_update(delta: float) virtual
int	mix_audio(num_frames: int, buffer: PackedFloat32Array = PackedFloat32Array(), offset: int = 0)

Method Descriptions

int **_get_channels**() virtual const 🔗

Returns the number of audio channels.

---

float **_get_length**() virtual const 🔗

Returns the video duration in seconds, if known, or 0 if unknown.

---

int **_get_mix_rate**() virtual const

Returns the audio sample rate used for mixing.

---

float **_get_playback_position**() virtual const

Return the current playback timestamp. Called in response to the VideoStreamPlayer.stream_position getter.

Texture2D **_get_texture**() virtual const

Allocates a Texture2D in which decoded video frames will be drawn.

---

bool **_is_paused**() virtual const

Returns the paused status, as set by _set_paused.

---

bool **_is_playing**() virtual const

Returns the playback state, as determined by calls to _play and _stop.

---

void **_play**() virtual

Called in response to VideoStreamPlayer.autoplay or VideoStreamPlayer.play. Note that manual playback may also invoke _stop multiple times before this method is called. _is_playing should return true once playing.

---

void **_seek**(time: float) virtual

Seeks to `time` seconds. Called in response to the VideoStreamPlayer.stream_position setter.

void **_set_audio_track**(idx: int) virtual

Select the audio track `idx`. Called when playback starts, and in response to the VideoStreamPlayer.audio_track setter.

void **_set_paused**(paused: bool) virtual

Set the paused status of video playback. _is_paused must return `paused`. Called in response to the VideoStreamPlayer.paused setter.

void **_stop**() virtual

Stops playback. May be called multiple times before _play, or in response to VideoStreamPlayer.stop. _is_playing should return false once stopped.

void **_update**(delta: float) virtual

Ticks video playback for `delta` seconds. Called every frame as long as _is_paused and _is_playing return true.

int **mix_audio**(num_frames: int, buffer: PackedFloat32Array = PackedFloat32Array(), offset: int = 0)

Render `num_frames` audio frames (of _get_channels floats each) from `buffer`, starting from index `offset` in the array. Returns the number of audio frames rendered, or -1 on error.

VideoStreamTheora

Inherits: VideoStream **<** Resource **<** RefCounted **<** Object

VideoStream resource for Ogg Theora videos.

Description

VideoStream resource handling the Ogg Theora [https://www.theora.org/] video format with `.ogv` extension. The Theora codec is decoded on the CPU.

Note: While Ogg Theora videos can also have an `.ogg` extension, you will have to rename the extension to `.ogv` to use those videos within Godot.

ViewportTexture

Inherits: Texture2D **<** Texture **<** Resource **<** RefCounted **<** Object

Provides the content of a Viewport as a dynamic texture.

Description

A **ViewportTexture** provides the content of a Viewport as a dynamic Texture2D. This can be used to combine the rendering of Control, Node2D and Node3D nodes. For example, you can use this texture to display a 3D scene inside a TextureRect, or a 2D overlay in a Sprite3D.

To get a **ViewportTexture** in code, use the Viewport.get_texture method on the target viewport.

Note: A **ViewportTexture** is always local to its scene (see Resource.resource_local_to_scene). If the scene root is not ready, it may return incorrect data (see Node.ready).

Note: Instantiating scenes containing a high-resolution **ViewportTexture** may cause noticeable stutter.

Tutorials

- GUI in 3D Viewport Demo [https://godotengine.org/asset-library/asset/2807]
- 3D in 2D Viewport Demo [https://godotengine.org/asset-library/asset/2804]
- 2D in 3D Viewport Demo [https://godotengine.org/asset-library/asset/2803]

- 3D Resolution Scaling Demo [https://godotengine.org/asset-library/asset/2805]

Properties

NodePath	viewport_path	`NodePath("")`

Property Descriptions

NodePath **viewport_path** = `NodePath("")` 🔗

- void **set_viewport_path_in_scene**(value: NodePath)
- NodePath **get_viewport_path_in_scene**()

The path to the Viewport node to display. This is relative to the local scene root (see Resource.get_local_scene), **not** to the nodes that use this texture.

Note: In the editor, this path is automatically updated when the target viewport or one of its ancestors is renamed or moved. At runtime, this path may not automatically update if the scene root cannot be found.

VisualShader

Inherits: Shader **<** Resource **<** RefCounted **<** Object

A custom shader program with a visual editor.

Description

This class provides a graph-like visual editor for creating a Shader. Although **VisualShader**s do not require coding, they share the same logic with script shaders. They use VisualShaderNodes that can be connected to each other to control the flow of the shader. The visual shader graph is converted to a script shader behind the scenes.

Tutorials

- Using VisualShaders

Properties

Vector2	graph_offset	`Vector2(0, 0)`

Methods

void	add_node(type: Type, node: VisualShaderNode, position: Vector2, id: int)

void	add_varying(name: String, mode: VaryingMode, type: VaryingType)
void	attach_node_to_frame(type: Type, id: int, frame: int)
bool	can_connect_nodes(type: Type, from_node: int, from_port: int, to_node: int, to_port: int) const
Error	connect_nodes(type: Type, from_node: int, from_port: int, to_node: int, to_port: int)
void	connect_nodes_forced(type: Type, from_node: int, from_port: int, to_node: int, to_port: int)
void	detach_node_from_frame(type: Type, id: int)
void	disconnect_nodes(type: Type, from_node: int, from_port: int, to_node: int, to_port: int)
VisualShaderNode	get_node(type: Type, id: int) const
Array[Dictionary]	get_node_connections(type: Type) const
PackedInt32Array	get_node_list(type: Type) const

Vector2	get_node_position(type: Type, id: int) const
int	get_valid_node_id(type: Type) const
bool	has_varying(name: String) const
bool	is_node_connection(type: Type, from_node: int, from_port: int, to_node: int, to_port: int) const
void	remove_node(type: Type, id: int)
void	remove_varying(name: String)
void	replace_node(type: Type, id: int, new_class: StringName)
void	set_mode(mode: Mode)
void	set_node_position(type: Type, id: int, position: Vector2)

Enumerations

enum **Type**: 🔗

Type **TYPE_VERTEX** = `0`

A vertex shader, operating on vertices.

Type **TYPE_FRAGMENT** = `1`

A fragment shader, operating on fragments (pixels).

Type **TYPE_LIGHT** = `2`

A shader for light calculations.

Type **TYPE_START** = `3`

A function for the "start" stage of particle shader.

Type **TYPE_PROCESS** = `4`

A function for the "process" stage of particle shader.

Type **TYPE_COLLIDE** = `5`

A function for the "collide" stage (particle collision handler) of particle shader.

Type **TYPE_START_CUSTOM** = `6`

A function for the "start" stage of particle shader, with customized output.

Type **TYPE_PROCESS_CUSTOM** = `7`

A function for the "process" stage of particle shader, with customized output.

Type **TYPE_SKY** = `8`

A shader for 3D environment's sky.

Type **TYPE_FOG** = `9`

A compute shader that runs for each froxel of the volumetric fog map.

Type **TYPE_MAX** = `10`

Represents the size of the Type enum.

---

enum **VaryingMode**: 🔗

VaryingMode **VARYING_MODE_VERTEX_TO_FRAG_LIGHT** = `0`

Varying is passed from `Vertex` function to `Fragment` and `Light` functions.

VaryingMode **VARYING_MODE_FRAG_TO_LIGHT** = `1`

Varying is passed from `Fragment` function to `Light` function.

VaryingMode **VARYING_MODE_MAX** = `2`

Represents the size of the VaryingMode enum.

---

enum **VaryingType**: 🔗

VaryingType **VARYING_TYPE_FLOAT** = `0`

Varying is of type float.

VaryingType **VARYING_TYPE_INT** = `1`

Varying is of type int.

VaryingType **VARYING_TYPE_UINT** = `2`

Varying is of type unsigned int.

VaryingType **VARYING_TYPE_VECTOR_2D** = `3`

Varying is of type Vector2.

VaryingType **VARYING_TYPE_VECTOR_3D** = `4`

Varying is of type Vector3.

VaryingType **VARYING_TYPE_VECTOR_4D** = `5`

Varying is of type Vector4.

VaryingType **VARYING_TYPE_BOOLEAN** = `6`

Varying is of type bool.

VaryingType **VARYING_TYPE_TRANSFORM** = `7`

Varying is of type Transform3D.

VaryingType **VARYING_TYPE_MAX** = `8`

Represents the size of the VaryingType enum.

Constants

NODE_ID_INVALID = `-1` 🔗

Indicates an invalid **VisualShader** node.

NODE_ID_OUTPUT = `0` 🔗

Indicates an output node of **VisualShader**.

Property Descriptions

Vector2 **graph_offset** = `Vector2(0, 0)` 🔗

- void **set_graph_offset**(value: Vector2)
- Vector2 **get_graph_offset**()

The offset vector of the whole graph.

Method Descriptions

void **add_node**(type: Type, node: VisualShaderNode, position: Vector2, id: int) 🔗

Adds the specified `node` to the shader.

---

void **add_varying**(name: String, mode: VaryingMode, type: VaryingType) 🔗

Adds a new varying value node to the shader.

---

void **attach_node_to_frame**(type: Type, id: int, frame: int) 🔗

Attaches the given node to the given frame.

---

bool **can_connect_nodes**(type: Type, from_node: int, from_port: int, to_node: int, to_port: int) const 🔗

Returns `true` if the specified nodes and ports can be connected together.

Error **connect_nodes**(type: Type, from_node: int, from_port: int, to_node: int, to_port: int) 🔗

Connects the specified nodes and ports.

void **connect_nodes_forced**(type: Type, from_node: int, from_port: int, to_node: int, to_port: int) 🔗

Connects the specified nodes and ports, even if they can't be connected. Such connection is invalid and will not function properly.

void **detach_node_from_frame**(type: Type, id: int) 🔗

Detaches the given node from the frame it is attached to.

void **disconnect_nodes**(type: Type, from_node: int, from_port: int, to_node: int, to_port: int) 🔗

Connects the specified nodes and ports.

VisualShaderNode **get_node**(type: Type, id: int) const 🔗

Returns the shader node instance with specified `type` and `id`.

Array[Dictionary] **get_node_connections**(type: Type) const 🔗

Returns the list of connected nodes with the specified type.

PackedInt32Array **get_node_list**(type: Type) const 🔗

Returns the list of all nodes in the shader with the specified type.

Vector2 **get_node_position**(type: Type, id: int) const 🔗

Returns the position of the specified node within the shader graph.

---

int **get_valid_node_id**(type: Type) const 🔗

Returns next valid node ID that can be added to the shader graph.

---

bool **has_varying**(name: String) const 🔗

Returns `true` if the shader has a varying with the given `name`.

---

bool **is_node_connection**(type: Type, from_node: int, from_port: int, to_node: int, to_port: int) const 🔗

Returns `true` if the specified node and port connection exist.

---

void **remove_node**(type: Type, id: int) 🔗

Removes the specified node from the shader.

---

void **remove_varying**(name: String) 🔗

Removes a varying value node with the given `name`. Prints an error if a node with this name is not found.

---

void **replace_node**(type: Type, id: int, new_class: StringName) 🔗

Replaces the specified node with a node of new class type.

---

void **set_mode**(mode: Mode) 🔗

Sets the mode of this shader.

void **set_node_position**(type: Type, id: int, position: Vector2) 🔗

Sets the position of the specified node.

VisualShaderNode

Inherits: Resource **<** RefCounted **<** Object

Inherited By: VisualShaderNodeBillboard, VisualShaderNodeClamp, VisualShaderNodeColorFunc, VisualShaderNodeColorOp, VisualShaderNodeCompare, VisualShaderNodeConstant, VisualShaderNodeCubemap, VisualShaderNodeCustom, VisualShaderNodeDerivativeFunc, VisualShaderNodeDeterminant, VisualShaderNodeDistanceFade, VisualShaderNodeDotProduct, VisualShaderNodeFloatFunc, VisualShaderNodeFloatOp, VisualShaderNodeFresnel, VisualShaderNodeIf, VisualShaderNodeInput, VisualShaderNodeIntFunc, VisualShaderNodeIntOp, VisualShaderNodeIs, VisualShaderNodeLinearSceneDepth, VisualShaderNodeMix, VisualShaderNodeMultiplyAdd, VisualShaderNodeOuterProduct, VisualShaderNodeOutput, VisualShaderNodeParameter, VisualShaderNodeParameterRef, VisualShaderNodeParticleAccelerator, VisualShaderNodeParticleConeVelocity, VisualShaderNodeParticleEmit, VisualShaderNodeParticleEmitter, VisualShaderNodeParticleMultiplyByAxisAngle, VisualShaderNodeParticleRandomness, VisualShaderNodeProximityFade, VisualShaderNodeRandomRange, VisualShaderNodeRemap, VisualShaderNodeReroute, VisualShaderNodeResizableBase, VisualShaderNodeRotationByAxis, VisualShaderNodeSample3D, VisualShaderNodeScreenNormalWorldSpace, VisualShaderNodeScreenUVToSDF, VisualShaderNodeSDFRaymarch, VisualShaderNodeSDFToScreenUV, VisualShaderNodeSmoothStep, VisualShaderNodeStep, VisualShaderNodeSwitch, VisualShaderNodeTexture, VisualShaderNodeTextureSDF, VisualShaderNodeTextureSDFNormal, VisualShaderNodeTransformCompose, VisualShaderNodeTransformDecompose, VisualShaderNodeTransformFunc, VisualShaderNodeTransformOp, VisualShaderNodeTransformVecMult, VisualShaderNodeUIntFunc, VisualShaderNodeUIntOp, VisualShaderNodeUVFunc, VisualShaderNodeUVPolarCoord, VisualShaderNodeVarying, VisualShaderNodeVectorBase, VisualShaderNodeWorldPositionFromDepth

Base class for VisualShader nodes. Not related to scene nodes.

Description

Visual shader graphs consist of various nodes. Each node in the graph is a separate object and they are represented as a rectangular boxes with title and a set of properties. Each node also has connection ports that allow to connect it to another nodes and control the flow of the shader.

Tutorials

- Using VisualShaders

Properties

int	linked_parent_graph_frame	`-1`
int	output_port_for_preview	`-1`

Methods

void	clear_default_input_values()
int	get_default_input_port(type: PortType) const
Array	get_default_input_values() const
Variant	get_input_port_default_value(port: int) const
void	remove_input_port_default_value(port: int)
void	set_default_input_values(values: Array)
void	set_input_port_default_value(port: int, value: Variant, prev_value: Variant = null)

Enumerations

enum **PortType**: 🔗

PortType **PORT_TYPE_SCALAR** = `0`

Floating-point scalar. Translated to `float` type in shader code.

PortType **PORT_TYPE_SCALAR_INT** = `1`

Integer scalar. Translated to `int` type in shader code.

PortType **PORT_TYPE_SCALAR_UINT** = `2`

Unsigned integer scalar. Translated to `uint` type in shader code.

PortType **PORT_TYPE_VECTOR_2D** = `3`

2D vector of floating-point values. Translated to `vec2` type in shader code.

PortType **PORT_TYPE_VECTOR_3D** = `4`

3D vector of floating-point values. Translated to `vec3` type in shader code.

PortType **PORT_TYPE_VECTOR_4D** = `5`

4D vector of floating-point values. Translated to `vec4` type in shader code.

PortType **PORT_TYPE_BOOLEAN** = `6`

Boolean type. Translated to `bool` type in shader code.

PortType **PORT_TYPE_TRANSFORM** = `7`

Transform type. Translated to `mat4` type in shader code.

PortType **PORT_TYPE_SAMPLER** = `8`

Sampler type. Translated to reference of sampler uniform in shader code. Can only be used for input ports in non-uniform nodes.

PortType **PORT_TYPE_MAX** = `9`

Represents the size of the PortType enum.

Property Descriptions

int **linked_parent_graph_frame** = `-1`

- void **set_frame**(value: int)
- int **get_frame**()

Represents the index of the frame this node is linked to. If set to `-1` the node is not linked to any frame.

---

int **output_port_for_preview** = `-1`

- void **set_output_port_for_preview**(value: int)
- int **get_output_port_for_preview**()

Sets the output port index which will be showed for preview. If set to `-1` no port will be open for preview.

Method Descriptions

void **clear_default_input_values**()

Clears the default input ports value.

---

int **get_default_input_port**(type: PortType) const

Returns the input port which should be connected by default when this node is created as a result of dragging a connection from an existing node to the empty space on the graph.

Array **get_default_input_values**() const

Returns an Array containing default values for all of the input ports of the node in the form `[index0, value0, index1, value1, ...]`.

Variant **get_input_port_default_value**(port: int) const

Returns the default value of the input `port`.

---

void **remove_input_port_default_value**(port: int)

Removes the default value of the input `port`.

---

void **set_default_input_values**(values: Array)

Sets the default input ports values using an Array of the form `[index0, value0, index1, value1, ...]`. For example: `[0, Vector3(0, 0, 0), 1, Vector3(0, 0, 0)]`.

---

void **set_input_port_default_value**(port: int, value: Variant, prev_value: Variant = null)

Sets the default `value` for the selected input `port`.

VisualShaderNodeBillboard

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A node that controls how the object faces the camera to be used within the visual shader graph.

Description

The output port of this node needs to be connected to `Model View Matrix` port of VisualShaderNodeOutput.

Properties

BillboardType	billboard_type	`1`
bool	keep_scale	`false`

Enumerations

enum **BillboardType**: 🔗

BillboardType **BILLBOARD_TYPE_DISABLED** = `0`

Billboarding is disabled and the node does nothing.

BillboardType **BILLBOARD_TYPE_ENABLED** = `1`

A standard billboarding algorithm is enabled.

BillboardType **BILLBOARD_TYPE_FIXED_Y** = `2`

A billboarding algorithm to rotate around Y-axis is enabled.

BillboardType **BILLBOARD_TYPE_PARTICLES** = `3`

A billboarding algorithm designed to use on particles is enabled.

BillboardType **BILLBOARD_TYPE_MAX** = `4`

Represents the size of the BillboardType enum.

Property Descriptions

BillboardType **billboard_type** = `1`

- void **set_billboard_type**(value: BillboardType)
- BillboardType **get_billboard_type**()

Controls how the object faces the camera. See BillboardType.

---

bool **keep_scale** = `false`

- void **set_keep_scale_enabled**(value: bool)
- bool **is_keep_scale_enabled**()

If `true`, the shader will keep the scale set for the mesh. Otherwise, the scale is lost when billboarding.

VisualShaderNodeBooleanConstant

Inherits: VisualShaderNodeConstant **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A boolean constant to be used within the visual shader graph.

Description

Has only one output port and no inputs.

Translated to `bool` in the shader language.

Properties

bool	constant	`false`

Property Descriptions

bool **constant** = `false`

- void **set_constant**(value: bool)
- bool **get_constant**()

A boolean constant which represents a state of this node.

VisualShaderNodeBooleanParameter

Inherits: VisualShaderNodeParameter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A boolean parameter to be used within the visual shader graph.

Description

Translated to `uniform bool` in the shader language.

Properties

bool	default_value	`false`
bool	default_value_enabled	`false`

Property Descriptions

bool **default_value** = `false` 🔗

- void **set_default_value**(value: bool)
- bool **get_default_value**()

A default value to be assigned within the shader.

---

bool **default_value_enabled** = `false` 🔗

- void **set_default_value_enabled**(value: bool)
- bool **is_default_value_enabled**()

Enables usage of the default_value.

VisualShaderNodeClamp

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Clamps a value within the visual shader graph.

Description

Constrains a value to lie between `min` and `max` values.

Properties

OpType	op_type	`0`

Enumerations

enum **OpType**: 🔗

OpType **OP_TYPE_FLOAT** = `0`

A floating-point scalar.

OpType **OP_TYPE_INT** = `1`

An integer scalar.

OpType **OP_TYPE_UINT** = `2`

An unsigned integer scalar.

OpType **OP_TYPE_VECTOR_2D** = `3`

A 2D vector type.

OpType **OP_TYPE_VECTOR_3D** = `4`

A 3D vector type.

OpType **OP_TYPE_VECTOR_4D** = `5`

A 4D vector type.

OpType **OP_TYPE_MAX** = `6`

Represents the size of the OpType enum.

Property Descriptions

OpType **op_type** = `0` 🔗

- void **set_op_type**(value: OpType)
- OpType **get_op_type**()

A type of operands and returned value.

VisualShaderNodeColorConstant

Inherits: VisualShaderNodeConstant **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A Color constant to be used within the visual shader graph.

Description

Has two output ports representing RGB and alpha channels of Color.

Translated to `vec3 rgb` and `float alpha` in the shader language.

Properties

Color	constant	`Color(1, 1, 1, 1)`

Property Descriptions

Color **constant** = `Color(1, 1, 1, 1)` 🔗

- void **set_constant**(value: Color)
- Color **get_constant**()

A Color constant which represents a state of this node.

VisualShaderNodeColorFunc

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A Color function to be used within the visual shader graph.

Description

Accept a Color to the input port and transform it according to function.

Properties

Function	function	`0`

Enumerations

enum **Function**: 🔗

Function **FUNC_GRAYSCALE** = `0`

Converts the color to grayscale using the following formula:

```
vec3 c = input;
float max1 = max(c.r, c.g);
float max2 = max(max1, c.b);
float max3 = max(max1, max2);
return vec3(max3, max3, max3);
```

Function **FUNC_HSV2RGB** = `1`

Converts HSV vector to RGB equivalent.

Function **FUNC_RGB2HSV** = `2`

Converts RGB vector to HSV equivalent.

Function **FUNC_SEPIA** = `3`

Applies sepia tone effect using the following formula:

```
vec3 c = input;
float r = (c.r * 0.393) + (c.g * 0.769) + (c.b * 0.189);
float g = (c.r * 0.349) + (c.g * 0.686) + (c.b * 0.168);
float b = (c.r * 0.272) + (c.g * 0.534) + (c.b * 0.131);
return vec3(r, g, b);
```

Function **FUNC_MAX** = `4`

Represents the size of the Function enum.

Property Descriptions

Function **function** = `0` 🔗

- void **set_function**(value: Function)
- Function **get_function**()

A function to be applied to the input color. See Function for options.

VisualShaderNodeColorOp

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A Color operator to be used within the visual shader graph.

Description

Applies operator to two color inputs.

Properties

Operator	operator	`0`

Enumerations

enum **Operator**: 🔗

Operator **OP_SCREEN** = `0`

Produce a screen effect with the following formula:

```
result = vec3(1.0) - (vec3(1.0) - a) * (vec3(1.0) - b);
```

Operator **OP_DIFFERENCE** = `1`

Produce a difference effect with the following formula:

```
result = abs(a - b);
```

Operator **OP_DARKEN** = 2

Produce a darken effect with the following formula:

```
result = min(a, b);
```

Operator **OP_LIGHTEN** = 3

Produce a lighten effect with the following formula:

```
result = max(a, b);
```

Operator **OP_OVERLAY** = 4

Produce an overlay effect with the following formula:

```
for (int i = 0; i < 3; i++) {
    float base = a[i];
    float blend = b[i];
    if (base < 0.5) {
        result[i] = 2.0 * base * blend;
    } else {
        result[i] = 1.0 - 2.0 * (1.0 - blend) * (1.0 - base);
    }
}
```

Operator **OP_DODGE** = 5

Produce a dodge effect with the following formula:

```
result = a / (vec3(1.0) - b);
```

Operator **OP_BURN** = 6

Produce a burn effect with the following formula:

```
result = vec3(1.0) - (vec3(1.0) - a) / b;
```

Operator **OP_SOFT_LIGHT** = 7

Produce a soft light effect with the following formula:

```
for (int i = 0; i < 3; i++) {
    float base = a[i];
    float blend = b[i];
    if (base < 0.5) {
        result[i] = base * (blend + 0.5);
    } else {
        result[i] = 1.0 - (1.0 - base) * (1.0 - (blend - 0.5));
    }
}
```

Operator **OP_HARD_LIGHT** = `8`

Produce a hard light effect with the following formula:

```
for (int i = 0; i < 3; i++) {
    float base = a[i];
    float blend = b[i];
    if (base < 0.5) {
        result[i] = base * (2.0 * blend);
    } else {
        result[i] = 1.0 - (1.0 - base) * (1.0 - 2.0 * (blend - 0.5));
    }
}
```

Operator **OP_MAX** = `9`

Represents the size of the Operator enum.

Property Descriptions

Operator **operator** = `0` 🔗

- void **set_operator**(value: Operator)
- Operator **get_operator**()

An operator to be applied to the inputs. See Operator for options.

VisualShaderNodeColorParameter

Inherits: VisualShaderNodeParameter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A Color parameter to be used within the visual shader graph.

Description

Translated to `uniform vec4` in the shader language.

Properties

Color	default_value	`Color(1, 1, 1, 1)`
bool	default_value_enabled	`false`

Property Descriptions

Color **default_value** = `Color(1, 1, 1, 1)`

- void **set_default_value**(value: Color)
- Color **get_default_value**()

A default value to be assigned within the shader.

---

bool **default_value_enabled** = `false` 🔗

- void **set_default_value_enabled**(value: bool)
- bool **is_default_value_enabled**()

Enables usage of the default_value.

VisualShaderNodeComment

Deprecated: This class has no function anymore and only exists for compatibility.

Inherits: VisualShaderNodeFrame **<** VisualShaderNodeResizableBase **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Only exists for compatibility. Use VisualShaderNodeFrame as a replacement.

Description

This node was replaced by VisualShaderNodeFrame and only exists to preserve compatibility. In the VisualShader editor it behaves exactly like VisualShaderNodeFrame.

Properties

String	description	`""`

Property Descriptions

String **description** = `""` 🔗

- void **set_description**(value: String)
- String **get_description**()

This property only exists to preserve data authored in earlier versions of Godot. It has currently no function.

VisualShaderNodeCompare

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A comparison function for common types within the visual shader graph.

Description

Compares `a` and `b` of type by function. Returns a boolean scalar. Translates to `if` instruction in shader code.

Properties

Condition	condition	`0`
Function	function	`0`
ComparisonType	type	`0`

Enumerations

enum **ComparisonType**: 🔗

ComparisonType **CTYPE_SCALAR** = `0`

A floating-point scalar.

ComparisonType **CTYPE_SCALAR_INT** = `1`

An integer scalar.

ComparisonType **CTYPE_SCALAR_UINT** = `2`

An unsigned integer scalar.

ComparisonType **CTYPE_VECTOR_2D** = `3`

A 2D vector type.

ComparisonType **CTYPE_VECTOR_3D** = `4`

A 3D vector type.

ComparisonType **CTYPE_VECTOR_4D** = `5`

A 4D vector type.

ComparisonType **CTYPE_BOOLEAN** = `6`

A boolean type.

ComparisonType **CTYPE_TRANSFORM** = `7`

A transform (`mat4`) type.

ComparisonType **CTYPE_MAX** = `8`

Represents the size of the ComparisonType enum.

---

enum **Function**: 🔗

Function **FUNC_EQUAL** = `0`

Comparison for equality (`a == b`).

Function **FUNC_NOT_EQUAL** = `1`

Comparison for inequality (`a != b`).

Function **FUNC_GREATER_THAN** = `2`

Comparison for greater than (`a > b`). Cannot be used if type set to CTYPE_BOOLEAN or CTYPE_TRANSFORM.

Function **FUNC_GREATER_THAN_EQUAL** = `3`

Comparison for greater than or equal (`a >= b`). Cannot be used if type set to CTYPE_BOOLEAN or CTYPE_TRANSFORM.

Function **FUNC_LESS_THAN** = `4`

Comparison for less than (`a < b`). Cannot be used if type set to CTYPE_BOOLEAN or CTYPE_TRANSFORM.

Function **FUNC_LESS_THAN_EQUAL** = `5`

Comparison for less than or equal (`a <= b`). Cannot be used if type set to CTYPE_BOOLEAN or CTYPE_TRANSFORM.

Function **FUNC_MAX** = `6`

Represents the size of the Function enum.

---

enum **Condition**: 🔗

Condition **COND_ALL** = `0`

The result will be true if all of component in vector satisfy the comparison condition.

Condition **COND_ANY** = `1`

The result will be true if any of component in vector satisfy the comparison condition.

Condition **COND_MAX** = `2`

Represents the size of the Condition enum.

Property Descriptions

Condition **condition** = `0` 🔗

- void **set_condition**(value: Condition)
- Condition **get_condition**()

Extra condition which is applied if type is set to CTYPE_VECTOR_3D.

Function **function** = `0` 🔗

- void **set_function**(value: Function)
- Function **get_function**()

A comparison function. See Function for options.

ComparisonType **type** = `0` 🔗

- void **set_comparison_type**(value: ComparisonType)
- ComparisonType **get_comparison_type**()

The type to be used in the comparison. See ComparisonType for options.

VisualShaderNodeConstant

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Inherited By: VisualShaderNodeBooleanConstant, VisualShaderNodeColorConstant, VisualShaderNodeFloatConstant, VisualShaderNodeIntConstant, VisualShaderNodeTransformConstant, VisualShaderNodeUIntConstant, VisualShaderNodeVec2Constant, VisualShaderNodeVec3Constant, VisualShaderNodeVec4Constant

A base type for the constants within the visual shader graph.

Description

This is an abstract class. See the derived types for descriptions of the possible values.

VisualShaderNodeCubemap

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A Cubemap sampling node to be used within the visual shader graph.

Description

Translated to `texture(cubemap, vec3)` in the shader language. Returns a color vector and alpha channel as scalar.

Properties

Cubemap	cube_map	
Source	source	`0`
TextureType	texture_type	`0`

Enumerations

enum **Source**: 🔗

Source **SOURCE_TEXTURE** = `0`

Use the Cubemap set via cube_map. If this is set to source, the `samplerCube` port is ignored.

Source **SOURCE_PORT** = `1`

Use the Cubemap sampler reference passed via the `samplerCube` port. If this is set to source, the cube_map texture is ignored.

Source **SOURCE_MAX** = `2`

Represents the size of the Source enum.

enum **TextureType**: 🔗

TextureType **TYPE_DATA** = `0`

No hints are added to the uniform declaration.

TextureType **TYPE_COLOR** = `1`

Adds `source_color` as hint to the uniform declaration for proper sRGB to linear conversion.

TextureType **TYPE_NORMAL_MAP** = `2`

Adds `hint_normal` as hint to the uniform declaration, which internally converts the texture for proper usage as normal map.

TextureType **TYPE_MAX** = `3`

Represents the size of the TextureType enum.

Property Descriptions

Cubemap **cube_map** 🔗

- void **set_cube_map**(value: Cubemap)
- Cubemap **get_cube_map**()

The Cubemap texture to sample when using SOURCE_TEXTURE as source.

Source **source** = 0 🔗

- void **set_source**(value: Source)
- Source **get_source**()

Defines which source should be used for the sampling. See Source for options.

TextureType **texture_type** = 0 🔗

- void **set_texture_type**(value: TextureType)
- TextureType **get_texture_type**()

Defines the type of data provided by the source texture. See TextureType for options.

VisualShaderNodeCubemapParameter

Inherits: VisualShaderNodeTextureParameter **<** VisualShaderNodeParameter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A Cubemap parameter node to be used within the visual shader graph.

Description

Translated to `uniform samplerCube` in the shader language. The output value can be used as port for VisualShaderNodeCubemap.

VisualShaderNodeCurveTexture

Inherits: VisualShaderNodeResizableBase **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Performs a CurveTexture lookup within the visual shader graph.

Description

Comes with a built-in editor for texture's curves.

Properties

CurveTexture	texture

Property Descriptions

CurveTexture **texture** 🔗

- void **set_texture**(value: CurveTexture)
- CurveTexture **get_texture**()

The source texture.

VisualShaderNodeCurveXYZTexture

Inherits: VisualShaderNodeResizableBase **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Performs a CurveXYZTexture lookup within the visual shader graph.

Description

Comes with a built-in editor for texture's curves.

Properties

CurveXYZTexture	texture

Property Descriptions

CurveXYZTexture **texture** 🔗

- void **set_texture**(value: CurveXYZTexture)
- CurveXYZTexture **get_texture**()

The source texture.

VisualShaderNodeCustom

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Virtual class to define custom VisualShaderNodes for use in the Visual Shader Editor.

Description

By inheriting this class you can create a custom VisualShader script addon which will be automatically added to the Visual Shader Editor. The VisualShaderNode's behavior is defined by overriding the provided virtual methods.

In order for the node to be registered as an editor addon, you must use the `@tool` annotation and provide a `class_name` for your custom script. For example:

```
@tool
extends VisualShaderNodeCustom
class_name VisualShaderNodeNoise
```

Tutorials

- Visual Shader plugins

Methods

String	_get_category() virtual const

String	_get_code(input_vars: Array[String], output_vars: Array[String], mode: Mode, type: Type) virtual const
int	_get_default_input_port(type: PortType) virtual const
String	_get_description() virtual const
String	_get_func_code(mode: Mode, type: Type) virtual const
String	_get_global_code(mode: Mode) virtual const
int	_get_input_port_count() virtual const
Variant	_get_input_port_default_value(port: int) virtual const
String	_get_input_port_name(port: int) virtual const
PortType	_get_input_port_type(port: int) virtual const
String	_get_name() virtual const
int	_get_output_port_count() virtual const

String	_get_output_port_name(port: int) virtual const
PortType	_get_output_port_type(port: int) virtual const
int	_get_property_count() virtual const
int	_get_property_default_index(index: int) virtual const
String	_get_property_name(index: int) virtual const
PackedStringArray	_get_property_options(index: int) virtual const
PortType	_get_return_icon_type() virtual const
bool	_is_available(mode: Mode, type: Type) virtual const
bool	_is_highend() virtual const
int	get_option_index(option: int) const

Method Descriptions

String **_get_category**() virtual const 🔗

Override this method to define the path to the associated custom node in the Visual Shader Editor's members dialog. The path may look like `"MyGame/MyFunctions/Noise"`.

Defining this method is **optional**. If not overridden, the node will be filed under the "Addons" category.

String **_get_code**(input_vars: Array[String], output_vars: Array[String], mode: Mode, type: Type) virtual const 🔗

Override this method to define the actual shader code of the associated custom node. The shader code should be returned as a string, which can have multiple lines (the `"""` multiline string construct can be used for convenience).

The `input_vars` and `output_vars` arrays contain the string names of the various input and output variables, as defined by `_get_input_*` and `_get_output_*` virtual methods in this class.

The output ports can be assigned values in the shader code. For example, `return output_vars[0] + " = " + input_vars[0] + ";"`.

You can customize the generated code based on the shader `mode` (see Mode) and/or `type` (see Type).

Defining this method is **required**.

---

int **_get_default_input_port**(type: PortType) virtual const 🔗

Override this method to define the input port which should be connected by default when this node is created as a result of dragging a connection from an existing node to the empty space on the graph.

Defining this method is **optional**. If not overridden, the connection will be created to the first valid port.

String **_get_description**() virtual const 🔗

Override this method to define the description of the associated custom node in the Visual Shader Editor's members dialog.

Defining this method is **optional**.

String **_get_func_code**(mode: Mode, type: Type) virtual const 🔗

Override this method to add a shader code to the beginning of each shader function (once). The shader code should be returned as a string, which can have multiple lines (the `"""` multiline string construct can be used for convenience).

If there are multiple custom nodes of different types which use this feature the order of each insertion is undefined.

You can customize the generated code based on the shader `mode` (see Mode) and/or `type` (see Type).

Defining this method is **optional**.

String **_get_global_code**(mode: Mode) virtual const 🔗

Override this method to add shader code on top of the global shader, to define your own standard library of reusable methods, varyings, constants, uniforms, etc. The shader code should be returned as a string, which can have multiple lines (the `"""` multiline string construct can be used for convenience).

Be careful with this functionality as it can cause name conflicts with other custom nodes, so be sure to give the defined entities unique names.

You can customize the generated code based on the shader `mode` (see Mode).

Defining this method is **optional**.

---

int **_get_input_port_count**() virtual const 🔗

Override this method to define the number of input ports of the associated custom node.

Defining this method is **required**. If not overridden, the node has no input ports.

Variant **_get_input_port_default_value**(port: int) virtual const 🔗

Override this method to define the default value for the specified input port. Prefer use this over VisualShaderNode.set_input_port_default_value.

Defining this method is **required**. If not overridden, the node has no default values for their input ports.

String **_get_input_port_name**(port: int) virtual const 🔗

Override this method to define the names of input ports of the associated custom node. The names are used both for the input slots in the editor and as identifiers in the shader code, and are passed in the `input_vars` array in _get_code.

Defining this method is **optional**, but recommended. If not overridden, input ports are named as `"in" + str(port)`.

PortType **_get_input_port_type**(port: int) virtual const

Override this method to define the returned type of each input port of the associated custom node (see PortType for possible types).

Defining this method is **optional**, but recommended. If not overridden, input ports will return the VisualShaderNode.PORT_TYPE_SCALAR type.

String **_get_name**() virtual const

Override this method to define the name of the associated custom node in the Visual Shader Editor's members dialog and graph.

Defining this method is **optional**, but recommended. If not overridden, the node will be named as "Unnamed".

int **_get_output_port_count**() virtual const

Override this method to define the number of output ports of the associated custom node.

Defining this method is **required**. If not overridden, the node has no output ports.

String **_get_output_port_name**(port: int) virtual const

Override this method to define the names of output ports of the associated custom node. The names are used both for the output slots in the editor and as identifiers in the shader code, and are passed in the `output_vars` array in _get_code.

Defining this method is **optional**, but recommended. If not overridden, output ports are named as `"out" + str(port)`.

PortType **_get_output_port_type**(port: int) virtual const 🔗

Override this method to define the returned type of each output port of the associated custom node (see PortType for possible types).

Defining this method is **optional**, but recommended. If not overridden, output ports will return the VisualShaderNode.PORT_TYPE_SCALAR type.

---

int **_get_property_count**() virtual const 🔗

Override this method to define the number of the properties.

Defining this method is **optional**.

---

int **_get_property_default_index**(index: int) virtual const 🔗

Override this method to define the default index of the property of the associated custom node.

Defining this method is **optional**.

String **_get_property_name**(index: int) virtual const 🔗

Override this method to define the names of the property of the associated custom node.

Defining this method is **optional**.

PackedStringArray **_get_property_options**(index: int) virtual const

Override this method to define the options inside the drop-down list property of the associated custom node.

Defining this method is **optional**.

PortType **_get_return_icon_type**() virtual const

Override this method to define the return icon of the associated custom node in the Visual Shader Editor's members dialog.

Defining this method is **optional**. If not overridden, no return icon is shown.

---

bool **_is_available**(mode: Mode, type: Type) virtual const

Override this method to prevent the node to be visible in the member dialog for the certain `mode` (see Mode) and/or `type` (see Type).

Defining this method is **optional**. If not overridden, it's `true`.

---

bool **_is_highend**() virtual const

Override this method to enable high-end mark in the Visual Shader Editor's members dialog.

Defining this method is **optional**. If not overridden, it's `false`.

---

int **get_option_index**(option: int) const 🔗

Returns the selected index of the drop-down list option within a graph. You may use this function to define the specific behavior in the _get_code or _get_global_code.

VisualShaderNodeDerivativeFunc

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Calculates a derivative within the visual shader graph.

Description

This node is only available in `Fragment` and `Light` visual shaders.

Properties

Function	function	`0`
OpType	op_type	`0`
Precision	precision	`0`

Enumerations

enum **OpType**: 🔗

OpType **OP_TYPE_SCALAR** = `0`

A floating-point scalar.

OpType **OP_TYPE_VECTOR_2D** = `1`

A 2D vector type.

OpType **OP_TYPE_VECTOR_3D** = `2`

A 3D vector type.

OpType **OP_TYPE_VECTOR_4D** = `3`

A 4D vector type.

OpType **OP_TYPE_MAX** = `4`

Represents the size of the OpType enum.

enum **Function**: 🔗

Function **FUNC_SUM** = `0`

Sum of absolute derivative in `x` and `y`.

Function **FUNC_X** = `1`

Derivative in `x` using local differencing.

Function **FUNC_Y** = `2`

Derivative in `y` using local differencing.

Function **FUNC_MAX** = `3`

Represents the size of the Function enum.

enum **Precision**: 🔗

Precision **PRECISION_NONE** = `0`

No precision is specified, the GPU driver is allowed to use whatever level of precision it chooses. This is the default option and is equivalent to using `dFdx()` or `dFdy()` in text shaders.

Precision **PRECISION_COARSE** = `1`

The derivative will be calculated using the current fragment's neighbors (which may not include the current fragment). This tends to be faster than using PRECISION_FINE, but may not be suitable when more precision is needed. This is equivalent to using `dFdxCoarse()` or `dFdyCoarse()` in text shaders.

Precision **PRECISION_FINE** = `2`

The derivative will be calculated using the current fragment and its immediate neighbors. This tends to be slower than using PRECISION_COARSE, but may be necessary when more precision is needed. This is equivalent to using `dFdxFine()` or `dFdyFine()` in text shaders.

Precision **PRECISION_MAX** = `3`

Represents the size of the Precision enum.

Property Descriptions

Function **function** = `0` 🔗

- void **set_function**(value: Function)
- Function **get_function**()

A derivative function type. See Function for options.

OpType **op_type** = 0 🔗

- void **set_op_type**(value: OpType)
- OpType **get_op_type**()

A type of operands and returned value. See OpType for options.

Precision **precision** = 0 🔗

- void **set_precision**(value: Precision)
- Precision **get_precision**()

Sets the level of precision to use for the derivative function. See Precision for options. When using the GL Compatibility renderer, this setting has no effect.

VisualShaderNodeDeterminant

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Calculates the determinant of a Transform3D within the visual shader graph.

Description

Translates to `determinant(x)` in the shader language.

VisualShaderNodeDistanceFade

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node representing distance fade effect.

Description

The distance fade effect fades out each pixel based on its distance to another object.

VisualShaderNodeDotProduct

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Calculates a dot product of two vectors within the visual shader graph.

Description

Translates to `dot(a, b)` in the shader language.

VisualShaderNodeExpression

Inherits: VisualShaderNodeGroupBase **<** VisualShaderNodeResizableBase **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Inherited By: VisualShaderNodeGlobalExpression

A custom visual shader graph expression written in Godot Shading Language.

Description

Custom Godot Shading Language expression, with a custom number of input and output ports.

The provided code is directly injected into the graph's matching shader function (`vertex`, `fragment`, or `light`), so it cannot be used to declare functions, varyings, uniforms, or global constants. See VisualShaderNodeGlobalExpression for such global definitions.

Properties

String	expression	`""`

Property Descriptions

String **expression** = `""` 🔗

- void **set_expression**(value: String)
- String **get_expression**()

An expression in Godot Shading Language, which will be injected at the start of the graph's matching shader function (`vertex`, `fragment`, or `light`), and thus cannot be used to declare functions, varyings, uniforms, or global constants.

VisualShaderNodeFaceForward

Inherits: VisualShaderNodeVectorBase **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Returns the vector that points in the same direction as a reference vector within the visual shader graph.

Description

Translates to `faceforward(N, I, Nref)` in the shader language. The function has three vector parameters: `N`, the vector to orient, `I`, the incident vector, and `Nref`, the reference vector. If the dot product of `I` and `Nref` is smaller than zero the return value is `N`. Otherwise, `-N` is returned.

VisualShaderNodeFloatConstant

Inherits: VisualShaderNodeConstant **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A scalar floating-point constant to be used within the visual shader graph.

Description

Translated to `float` in the shader language.

Properties

float	constant	`0.0`

Property Descriptions

float **constant** = `0.0` 🔗

- void **set_constant**(value: float)
- float **get_constant**()

A floating-point constant which represents a state of this node.

VisualShaderNodeFloatFunc

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A scalar floating-point function to be used within the visual shader graph.

Description

Accept a floating-point scalar (`x`) to the input port and transform it according to function.

Properties

Function	function	`13`

Enumerations

enum **Function**: 🔗

Function **FUNC_SIN** = `0`

Returns the sine of the parameter. Translates to `sin(x)` in the Godot Shader Language.

Function **FUNC_COS** = `1`

Returns the cosine of the parameter. Translates to `cos(x)` in the Godot Shader Language.

Function **FUNC_TAN** = `2`

Returns the tangent of the parameter. Translates to `tan(x)` in the Godot Shader Language.

Function **FUNC_ASIN** = `3`

Returns the arc-sine of the parameter. Translates to `asin(x)` in the Godot Shader Language.

Function **FUNC_ACOS** = `4`

Returns the arc-cosine of the parameter. Translates to `acos(x)` in the Godot Shader Language.

Function **FUNC_ATAN** = `5`

Returns the arc-tangent of the parameter. Translates to `atan(x)` in the Godot Shader Language.

Function **FUNC_SINH** = `6`

Returns the hyperbolic sine of the parameter. Translates to `sinh(x)` in the Godot Shader Language.

Function **FUNC_COSH** = `7`

Returns the hyperbolic cosine of the parameter. Translates to `cosh(x)` in the Godot Shader Language.

Function **FUNC_TANH** = `8`

Returns the hyperbolic tangent of the parameter. Translates to `tanh(x)` in the Godot Shader Language.

Function **FUNC_LOG** = `9`

Returns the natural logarithm of the parameter. Translates to `log(x)` in the Godot Shader Language.

Function **FUNC_EXP** = `10`

Returns the natural exponentiation of the parameter. Translates to `exp(x)` in the Godot Shader Language.

Function **FUNC_SQRT** = `11`

Returns the square root of the parameter. Translates to `sqrt(x)` in the Godot Shader Language.

Function **FUNC_ABS** = `12`

Returns the absolute value of the parameter. Translates to `abs(x)` in the Godot Shader Language.

Function **FUNC_SIGN** = `13`

Extracts the sign of the parameter. Translates to `sign(x)` in the Godot Shader Language.

Function **FUNC_FLOOR** = `14`

Finds the nearest integer less than or equal to the parameter. Translates to `floor(x)` in the Godot Shader Language.

Function **FUNC_ROUND** = `15`

Finds the nearest integer to the parameter. Translates to `round(x)` in the Godot Shader Language.

Function **FUNC_CEIL** = `16`

Finds the nearest integer that is greater than or equal to the parameter. Translates to `ceil(x)` in the Godot Shader Language.

Function **FUNC_FRACT** = `17`

Computes the fractional part of the argument. Translates to `fract(x)` in the Godot Shader Language.

Function **FUNC_SATURATE** = `18`

Clamps the value between `0.0` and `1.0` using `min(max(x, 0.0), 1.0)`.

Function **FUNC_NEGATE** = `19`

Negates the `x` using `-(x)`.

Function **FUNC_ACOSH** = `20`

Returns the arc-hyperbolic-cosine of the parameter. Translates to `acosh(x)` in the Godot Shader Language.

Function **FUNC_ASINH** = `21`

Returns the arc-hyperbolic-sine of the parameter. Translates to `asinh(x)` in the Godot Shader Language.

Function **FUNC_ATANH** = `22`

Returns the arc-hyperbolic-tangent of the parameter. Translates to `atanh(x)` in the Godot Shader Language.

Function **FUNC_DEGREES** = `23`

Convert a quantity in radians to degrees. Translates to `degrees(x)` in the Godot Shader Language.

Function **FUNC_EXP2** = `24`

Returns 2 raised by the power of the parameter. Translates to `exp2(x)` in the Godot Shader Language.

Function **FUNC_INVERSE_SQRT** = `25`

Returns the inverse of the square root of the parameter. Translates to `inversesqrt(x)` in the Godot Shader Language.

Function **FUNC_LOG2** = `26`

Returns the base 2 logarithm of the parameter. Translates to `log2(x)` in the Godot Shader Language.

Function **FUNC_RADIANS** = `27`

Convert a quantity in degrees to radians. Translates to `radians(x)` in the Godot Shader Language.

Function **FUNC_RECIPROCAL** = `28`

Finds reciprocal value of dividing 1 by `x` (i.e. `1 / x`).

Function **FUNC_ROUNDEVEN** = `29`

Finds the nearest even integer to the parameter. Translates to `roundEven(x)` in the Godot Shader Language.

Function **FUNC_TRUNC** = `30`

Returns a value equal to the nearest integer to `x` whose absolute value is not larger than the absolute value of `x`. Translates to `trunc(x)` in the Godot Shader Language.

Function **FUNC_ONEMINUS** = `31`

Subtracts scalar `x` from 1 (i.e. `1 - x`).

Function **FUNC_MAX** = `32`

Represents the size of the Function enum.

Property Descriptions

Function **function** = `13`

- void **set_function**(value: Function)
- Function **get_function**()

A function to be applied to the scalar. See Function for options.

VisualShaderNodeFloatOp

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A floating-point scalar operator to be used within the visual shader graph.

Description

Applies operator to two floating-point inputs: `a` and `b`.

Properties

Operator	operator	`0`

Enumerations

enum **Operator**: 🔗

Operator **OP_ADD** = `0`

Sums two numbers using `a + b`.

Operator **OP_SUB** = `1`

Subtracts two numbers using `a - b`.

Operator **OP_MUL** = `2`

Multiplies two numbers using `a * b`.

Operator **OP_DIV** = `3`

Divides two numbers using `a / b`.

Operator **OP_MOD** = `4`

Calculates the remainder of two numbers. Translates to `mod(a, b)` in the Godot Shader Language.

Operator **OP_POW** = `5`

Raises the `a` to the power of `b`. Translates to `pow(a, b)` in the Godot Shader Language.

Operator **OP_MAX** = `6`

Returns the greater of two numbers. Translates to `max(a, b)` in the Godot Shader Language.

Operator **OP_MIN** = `7`

Returns the lesser of two numbers. Translates to `min(a, b)` in the Godot Shader Language.

Operator **OP_ATAN2** = `8`

Returns the arc-tangent of the parameters. Translates to `atan(a, b)` in the Godot Shader Language.

Operator **OP_STEP** = `9`

Generates a step function by comparing `b`(x) to `a`(edge). Returns 0.0 if `x` is smaller than `edge` and otherwise 1.0. Translates to `step(a, b)` in the Godot Shader Language.

Operator **OP_ENUM_SIZE** = `10`

Represents the size of the Operator enum.

Property Descriptions

Operator **operator** = `0` 🔗

- void **set_operator**(value: Operator)
- Operator **get_operator**()

An operator to be applied to the inputs. See Operator for options.

VisualShaderNodeFloatParameter

Inherits: VisualShaderNodeParameter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A scalar float parameter to be used within the visual shader graph.

Description

Translated to `uniform float` in the shader language.

Properties

float	default_value	`0.0`
bool	default_value_enabled	`false`
Hint	hint	`0`
float	max	`1.0`
float	min	`0.0`
float	step	`0.1`

Enumerations

enum **Hint**: 🔗

Hint **HINT_NONE** = `0`

No hint used.

Hint **HINT_RANGE** = `1`

A range hint for scalar value, which limits possible input values between min and max. Translated to `hint_range(min, max)` in shader code.

Hint **HINT_RANGE_STEP** = `2`

A range hint for scalar value with step, which limits possible input values between min and max, with a step (increment) of step. Translated to `hint_range(min, max, step)` in shader code.

Hint **HINT_MAX** = `3`

Represents the size of the Hint enum.

Property Descriptions

float **default_value** = `0.0` 🔗

- void **set_default_value**(value: float)
- float **get_default_value**()

A default value to be assigned within the shader.

---

bool **default_value_enabled** = `false` 🔗

- void **set_default_value_enabled**(value: bool)
- bool **is_default_value_enabled**()

Enables usage of the default_value.

Hint **hint** = `0` 🔗

- void **set_hint**(value: Hint)
- Hint **get_hint**()

A hint applied to the uniform, which controls the values it can take when set through the Inspector.

---

float **max** = `1.0` 🔗

- void **set_max**(value: float)
- float **get_max**()

Minimum value for range hints. Used if hint is set to HINT_RANGE or HINT_RANGE_STEP.

---

float **min** = `0.0` 🔗

- void **set_min**(value: float)
- float **get_min**()

Maximum value for range hints. Used if hint is set to HINT_RANGE or HINT_RANGE_STEP.

---

float **step** = `0.1` 🔗

- void **set_step**(value: float)
- float **get_step**()

Step (increment) value for the range hint with step. Used if hint is set to HINT_RANGE_STEP.

VisualShaderNodeFrame

Inherits: VisualShaderNodeResizableBase **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Inherited By: VisualShaderNodeComment

A frame other visual shader nodes can be attached to for better organization.

Description

A rectangular frame that can be used to group visual shader nodes together to improve organization.

Nodes attached to the frame will move with it when it is dragged and it can automatically resize to enclose all attached nodes.

Its title, description and color can be customized.

Properties

PackedInt32Array	attached_nodes	`PackedInt32Array()`
bool	autoshrink	`true`
Color	tint_color	`Color(0.3, 0.3, 0.3, 0.75)`
bool	tint_color_enabled	`false`

String	title	`"Title"`

Methods

void	add_attached_node(node: int)
void	remove_attached_node(node: int)

Property Descriptions

PackedInt32Array **attached_nodes** = `PackedInt32Array()` 🔗

- void **set_attached_nodes**(value: PackedInt32Array)
- PackedInt32Array **get_attached_nodes**()

The list of nodes attached to the frame.

Note: The returned array is *copied* and any changes to it will not update the original property value. See PackedInt32Array for more details.

---

bool **autoshrink** = `true` 🔗

- void **set_autoshrink_enabled**(value: bool)
- bool **is_autoshrink_enabled**()

If `true`, the frame will automatically resize to enclose all attached nodes.

Color **tint_color** = `Color(0.3, 0.3, 0.3, 0.75)` 🔗

- void **set_tint_color**(value: Color)
- Color **get_tint_color**()

The color of the frame when tint_color_enabled is `true`.

---

bool **tint_color_enabled** = `false` 🔗

- void **set_tint_color_enabled**(value: bool)
- bool **is_tint_color_enabled**()

If `true`, the frame will be tinted with the color specified in tint_color.

String **title** = `"Title"` 🔗

- void **set_title**(value: String)
- String **get_title**()

The title of the node.

Method Descriptions

void **add_attached_node**(node: int) 🔗

Adds a node to the list of nodes attached to the frame. Should not be called directly, use the VisualShader.attach_node_to_frame method instead.

---

void **remove_attached_node**(node: int) 🔗

Removes a node from the list of nodes attached to the frame. Should not be called directly, use the VisualShader.detach_node_from_frame method instead.

VisualShaderNodeFresnel

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A Fresnel effect to be used within the visual shader graph.

Description

Returns falloff based on the dot product of surface normal and view direction of camera (pass associated inputs to it).

VisualShaderNodeGlobalExpression

Inherits: VisualShaderNodeExpression **<** VisualShaderNodeGroupBase **<** VisualShaderNodeResizableBase **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A custom global visual shader graph expression written in Godot Shading Language.

Description

Custom Godot Shader Language expression, which is placed on top of the generated shader. You can place various function definitions inside to call later in VisualShaderNodeExpressions (which are injected in the main shader functions). You can also declare varyings, uniforms and global constants.

VisualShaderNodeGroupBase

Inherits: VisualShaderNodeResizableBase **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Inherited By: VisualShaderNodeExpression

Base class for a family of nodes with variable number of input and output ports within the visual shader graph.

Description

Currently, has no direct usage, use the derived classes instead.

Methods

void	add_input_port(id: int, type: int, name: String)
void	add_output_port(id: int, type: int, name: String)
void	clear_input_ports()
void	clear_output_ports()
int	get_free_input_port_id() const
int	get_free_output_port_id() const

int	get_input_port_count() const
String	get_inputs() const
int	get_output_port_count() const
String	get_outputs() const
bool	has_input_port(id: int) const
bool	has_output_port(id: int) const
bool	is_valid_port_name(name: String) const
void	remove_input_port(id: int)
void	remove_output_port(id: int)
void	set_input_port_name(id: int, name: String)
void	set_input_port_type(id: int, type: int)
void	set_inputs(inputs: String)
void	set_output_port_name(id: int, name: String)

void	set_output_port_type(id: int, type: int)
void	set_outputs(outputs: String)

Method Descriptions

void **add_input_port**(id: int, type: int, name: String) 🔗

Adds an input port with the specified `type` (see PortType) and `name`.

void **add_output_port**(id: int, type: int, name: String) 🔗

Adds an output port with the specified `type` (see PortType) and `name`.

void **clear_input_ports**() 🔗

Removes all previously specified input ports.

void **clear_output_ports**() 🔗

Removes all previously specified output ports.

int **get_free_input_port_id**() const 🔗

Returns a free input port ID which can be used in add_input_port.

int **get_free_output_port_id**() const

Returns a free output port ID which can be used in add_output_port.

---

int **get_input_port_count**() const

Returns the number of input ports in use. Alternative for get_free_input_port_id.

String **get_inputs**() const

Returns a String description of the input ports as a colon-separated list using the format `id,type,name;` (see add_input_port).

---

int **get_output_port_count**() const

Returns the number of output ports in use. Alternative for get_free_output_port_id.

String **get_outputs**() const

Returns a String description of the output ports as a colon-separated list using the format `id,type,name;` (see add_output_port).

---

bool **has_input_port**(id: int) const

Returns `true` if the specified input port exists.

---

bool **has_output_port**(id: int) const

Returns `true` if the specified output port exists.

---

bool **is_valid_port_name**(name: String) const

Returns `true` if the specified port name does not override an existed port name and is valid within the shader.

---

void **remove_input_port**(id: int)

Removes the specified input port.

---

void **remove_output_port**(id: int)

Removes the specified output port.

---

void **set_input_port_name**(id: int, name: String)

Renames the specified input port.

---

void **set_input_port_type**(id: int, type: int)

Sets the specified input port's type (see PortType).

---

void **set_inputs**(inputs: String)

Defines all input ports using a String formatted as a colon-separated list: `id,type,name;` (see add_input_port).

---

void **set_output_port_name**(id: int, name: String)

Renames the specified output port.

---

void **set_output_port_type**(id: int, type: int)

Sets the specified output port's type (see PortType).

---

void **set_outputs**(outputs: String)

Defines all output ports using a String formatted as a colon-separated list: `id,type,name;` (see add_output_port).

VisualShaderNodeIf

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Outputs a 3D vector based on the result of a floating-point comparison within the visual shader graph.

Description

This visual shader node has six input ports:

- Port **1** and **2** provide the two floating-point numbers `a` and `b` that will be compared.
- Port **3** is the tolerance, which allows similar floating-point numbers to be considered equal.
- Ports **4**, **5**, and **6** are the possible outputs, returned if `a == b`, `a > b`, or `a < b` respectively.

VisualShaderNodeInput

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Represents the input shader parameter within the visual shader graph.

Description

Gives access to input variables (built-ins) available for the shader. See the shading reference for the list of available built-ins for each shader type (check `Tutorials` section for link).

Tutorials

- Shading reference index

Properties

String	input_name	`"[None]"`

Methods

String	get_input_real_name() const

Signals

input_type_changed()

Emitted when input is changed via input_name.

Property Descriptions

String **input_name** = `"[None]"`

- void **set_input_name**(value: String)
- String **get_input_name**()

One of the several input constants in lower-case style like: "vertex" (`VERTEX`) or "point_size" (`POINT_SIZE`).

Method Descriptions

String **get_input_real_name**() const

Returns a translated name of the current constant in the Godot Shader Language. E.g. `"ALBEDO"` if the input_name equal to `"albedo"`.

VisualShaderNodeIntConstant

Inherits: VisualShaderNodeConstant **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A scalar integer constant to be used within the visual shader graph.

Description

Translated to `int` in the shader language.

Properties

int	constant	`0`

Property Descriptions

int **constant** = `0` 🔗

- void **set_constant**(value: int)
- int **get_constant**()

An integer constant which represents a state of this node.

VisualShaderNodeIntFunc

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A scalar integer function to be used within the visual shader graph.

Description

Accept an integer scalar (`x`) to the input port and transform it according to function.

Properties

Function	function	`2`

Enumerations

enum **Function**: 🔗

Function **FUNC_ABS** = `0`

Returns the absolute value of the parameter. Translates to `abs(x)` in the Godot Shader Language.

Function **FUNC_NEGATE** = `1`

Negates the `x` using `-(x)`.

Function **FUNC_SIGN** = `2`

Extracts the sign of the parameter. Translates to `sign(x)` in the Godot Shader Language.

Function **FUNC_BITWISE_NOT** = `3`

Returns the result of bitwise `NOT` operation on the integer. Translates to `~a` in the Godot Shader Language.

Function **FUNC_MAX** = `4`

Represents the size of the Function enum.

Property Descriptions

Function **function** = `2` 🔗

- void **set_function**(value: Function)
- Function **get_function**()

A function to be applied to the scalar. See Function for options.

VisualShaderNodeIntOp

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

An integer scalar operator to be used within the visual shader graph.

Description

Applies operator to two integer inputs: `a` and `b`.

Properties

Operator	operator	`0`

Enumerations

enum **Operator**: 🔗

Operator **OP_ADD** = `0`

Sums two numbers using `a + b`.

Operator **OP_SUB** = `1`

Subtracts two numbers using `a - b`.

Operator **OP_MUL** = `2`

Multiplies two numbers using `a * b`.

[Operator](#) **OP_DIV** = `3`

Divides two numbers using `a / b`.

[Operator](#) **OP_MOD** = `4`

Calculates the remainder of two numbers using `a % b`.

[Operator](#) **OP_MAX** = `5`

Returns the greater of two numbers. Translates to `max(a, b)` in the Godot Shader Language.

[Operator](#) **OP_MIN** = `6`

Returns the lesser of two numbers. Translates to `max(a, b)` in the Godot Shader Language.

[Operator](#) **OP_BITWISE_AND** = `7`

Returns the result of bitwise `AND` operation on the integer. Translates to `a & b` in the Godot Shader Language.

[Operator](#) **OP_BITWISE_OR** = `8`

Returns the result of bitwise `OR` operation for two integers. Translates to `a | b` in the Godot Shader Language.

[Operator](#) **OP_BITWISE_XOR** = `9`

Returns the result of bitwise `XOR` operation for two integers. Translates to `a ^ b` in the Godot Shader Language.

[Operator](#) **OP_BITWISE_LEFT_SHIFT** = `10`

Returns the result of bitwise left shift operation on the integer. Translates to `a << b` in the Godot Shader Language.

Operator **OP_BITWISE_RIGHT_SHIFT** = `11`

Returns the result of bitwise right shift operation on the integer. Translates to `a >> b` in the Godot Shader Language.

Operator **OP_ENUM_SIZE** = `12`

Represents the size of the Operator enum.

Property Descriptions

Operator **operator** = `0`

- void **set_operator**(value: Operator)
- Operator **get_operator**()

An operator to be applied to the inputs. See Operator for options.

VisualShaderNodeIntParameter

Inherits: VisualShaderNodeParameter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node for shader parameter (uniform) of type int.

Description

A VisualShaderNodeParameter of type int. Offers additional customization for range of accepted values.

Properties

int	default_value	`0`
bool	default_value_enabled	`false`
Hint	hint	`0`
int	max	`100`
int	min	`0`

int	step	`1`

Enumerations

enum **Hint**: 🔗

Hint **HINT_NONE** = `0`

The parameter will not constrain its value.

Hint **HINT_RANGE** = `1`

The parameter's value must be within the specified min/max range.

Hint **HINT_RANGE_STEP** = `2`

The parameter's value must be within the specified range, with the given step between values.

Hint **HINT_MAX** = `3`

Represents the size of the Hint enum.

Property Descriptions

int **default_value** = `0` 🔗

- void **set_default_value**(value: int)
- int **get_default_value**()

Default value of this parameter, which will be used if not set externally. default_value_enabled must be enabled; defaults to `0` otherwise.

---

bool **default_value_enabled** = `false` 🔗

- void **set_default_value_enabled**(value: bool)
- bool **is_default_value_enabled**()

If `true`, the node will have a custom default value.

Hint **hint** = `0` 🔗

- void **set_hint**(value: Hint)
- Hint **get_hint**()

Range hint of this node. Use it to customize valid parameter range.

---

int **max** = `100` 🔗

- void **set_max**(value: int)
- int **get_max**()

The maximum value this parameter can take. hint must be either HINT_RANGE or HINT_RANGE_STEP for this to take effect.

---

int **min** = `0` 🔗

- void **set_min**(value: int)
- int **get_min**()

The minimum value this parameter can take. hint must be either HINT_RANGE or HINT_RANGE_STEP for this to take effect.

---

int **step** = `1` 🔗

- void **set_step**(value: int)
- int **get_step**()

The step between parameter's values. Forces the parameter to be a multiple of the given value. hint must be HINT_RANGE_STEP for this to take effect.

VisualShaderNodeIs

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A boolean comparison operator to be used within the visual shader graph.

Description

Returns the boolean result of the comparison between `INF` or `NaN` and a scalar parameter.

Properties

Function	function	`0`

Enumerations

enum **Function**: 🔗

Function **FUNC_IS_INF** = `0`

Comparison with `INF` (Infinity).

Function **FUNC_IS_NAN** = `1`

Comparison with `NaN` (Not a Number; indicates invalid numeric results, such as division by zero).

Function **FUNC_MAX** = `2`

Represents the size of the Function enum.

Property Descriptions

Function **function** = `0`

- void **set_function**(value: Function)
- Function **get_function**()

The comparison function. See Function for options.

VisualShaderNodeLinearSceneDepth

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node that returns the depth value of the DEPTH_TEXTURE node in a linear space.

Description

This node can be used in fragment shaders.

VisualShaderNodeMix

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Linearly interpolates between two values within the visual shader graph.

Description

Translates to `mix(a, b, weight)` in the shader language.

Properties

OpType	op_type	`0`

Enumerations

enum **OpType**: 🔗

OpType **OP_TYPE_SCALAR** = `0`

A floating-point scalar.

OpType **OP_TYPE_VECTOR_2D** = `1`

A 2D vector type.

OpType **OP_TYPE_VECTOR_2D_SCALAR** = `2`

The `a` and `b` ports use a 2D vector type. The `weight` port uses a scalar type.

OpType **OP_TYPE_VECTOR_3D** = `3`

A 3D vector type.

OpType **OP_TYPE_VECTOR_3D_SCALAR** = `4`

The `a` and `b` ports use a 3D vector type. The `weight` port uses a scalar type.

OpType **OP_TYPE_VECTOR_4D** = `5`

A 4D vector type.

OpType **OP_TYPE_VECTOR_4D_SCALAR** = `6`

The `a` and `b` ports use a 4D vector type. The `weight` port uses a scalar type.

OpType **OP_TYPE_MAX** = `7`

Represents the size of the OpType enum.

Property Descriptions

OpType **op_type** = `0` 🔗

- void **set_op_type**(value: OpType)
- OpType **get_op_type**()

A type of operands and returned value.

VisualShaderNodeMultiplyAdd

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Performs a fused multiply-add operation within the visual shader graph.

Description

Uses three operands to compute `(a * b + c)` expression.

Properties

OpType	op_type	`0`

Enumerations

enum **OpType**: 🔗

OpType **OP_TYPE_SCALAR** = `0`

A floating-point scalar type.

OpType **OP_TYPE_VECTOR_2D** = `1`

A 2D vector type.

OpType **OP_TYPE_VECTOR_3D** = `2`

A 3D vector type.

OpType **OP_TYPE_VECTOR_4D** = `3`

A 4D vector type.

OpType **OP_TYPE_MAX** = `4`

Represents the size of the OpType enum.

Property Descriptions

OpType **op_type** = `0`

- void **set_op_type**(value: OpType)
- OpType **get_op_type**()

A type of operands and returned value.

VisualShaderNodeOuterProduct

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Calculates an outer product of two vectors within the visual shader graph.

Description

`OuterProduct` treats the first parameter `c` as a column vector (matrix with one column) and the second parameter `r` as a row vector (matrix with one row) and does a linear algebraic matrix multiply `c * r`, yielding a matrix whose number of rows is the number of components in `c` and whose number of columns is the number of components in `r`.

VisualShaderNodeOutput

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Inherited By: VisualShaderNodeParticleOutput

Represents the output shader parameters within the visual shader graph.

Description

This visual shader node is present in all shader graphs in form of "Output" block with multiple output value ports.

VisualShaderNodeParameter

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Inherited By: VisualShaderNodeBooleanParameter, VisualShaderNodeColorParameter, VisualShaderNodeFloatParameter, VisualShaderNodeIntParameter, VisualShaderNodeTextureParameter, VisualShaderNodeTransformParameter, VisualShaderNodeUIntParameter, VisualShaderNodeVec2Parameter, VisualShaderNodeVec3Parameter, VisualShaderNodeVec4Parameter

A base type for the parameters within the visual shader graph.

Description

A parameter represents a variable in the shader which is set externally, i.e. from the ShaderMaterial. Parameters are exposed as properties in the ShaderMaterial and can be assigned from the Inspector or from a script.

Properties

String	parameter_name	`""`
Qualifier	qualifier	`0`

Enumerations

enum **Qualifier**:

Qualifier **QUAL_NONE** = `0`

The parameter will be tied to the ShaderMaterial using this shader.

Qualifier **QUAL_GLOBAL** = `1`

The parameter will use a global value, defined in Project Settings.

Qualifier **QUAL_INSTANCE** = `2`

The parameter will be tied to the node with attached ShaderMaterial using this shader.

Qualifier **QUAL_MAX** = `3`

Represents the size of the Qualifier enum.

Property Descriptions

String **parameter_name** = `""`

- void **set_parameter_name**(value: String)
- String **get_parameter_name**()

Name of the parameter, by which it can be accessed through the ShaderMaterial properties.

Qualifier **qualifier** = `0`

- void **set_qualifier**(value: Qualifier)
- Qualifier **get_qualifier**()

Defines the scope of the parameter.

VisualShaderNodeParameterRef

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A reference to an existing VisualShaderNodeParameter.

Description

Creating a reference to a VisualShaderNodeParameter allows you to reuse this parameter in different shaders or shader stages easily.

Properties

String	parameter_name	`"[None]"`

Property Descriptions

String **parameter_name** = `"[None]"` 🔗

- void **set_parameter_name**(value: String)
- String **get_parameter_name**()

The name of the parameter which this reference points to.

VisualShaderNodeParticleAccelerator

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node that accelerates particles.

Description

Particle accelerator can be used in "process" step of particle shader. It will accelerate the particles. Connect it to the Velocity output port.

Properties

Mode	mode	`0`

Enumerations

enum **Mode**: 🔗

Mode **MODE_LINEAR** = `0`

The particles will be accelerated based on their velocity.

Mode **MODE_RADIAL** = `1`

The particles will be accelerated towards or away from the center.

Mode **MODE_TANGENTIAL** = `2`

The particles will be accelerated tangentially to the radius vector from center to their position.

Mode **MODE_MAX** = `3`

Represents the size of the Mode enum.

Property Descriptions

Mode **mode** = `0` 🔗

- void **set_mode**(value: Mode)
- Mode **get_mode**()

Defines in what manner the particles will be accelerated.

VisualShaderNodeParticleBoxEmitter

Inherits: VisualShaderNodeParticleEmitter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node that makes particles emitted in a box shape.

Description

VisualShaderNodeParticleEmitter that makes the particles emitted in box shape with the specified extents.

VisualShaderNodeParticleConeVelocity

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node that makes particles move in a cone shape.

Description

This node can be used in "start" step of particle shader. It defines the initial velocity of the particles, making them move in cone shape starting from the center, with a given spread.

VisualShaderNodeParticleEmit

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node that forces to emit a particle from a sub-emitter.

Description

This node internally calls `emit_subparticle` shader method. It will emit a particle from the configured sub-emitter and also allows to customize how its emitted. Requires a sub-emitter assigned to the particles node with this shader.

Properties

EmitFlags	flags	`31`

Enumerations

enum **EmitFlags**: 🔗

EmitFlags **EMIT_FLAG_POSITION** = `1`

If enabled, the particle starts with the position defined by this node.

EmitFlags **EMIT_FLAG_ROT_SCALE** = `2`

If enabled, the particle starts with the rotation and scale defined by this node.

EmitFlags **EMIT_FLAG_VELOCITY** = `4`

If enabled,the particle starts with the velocity defined by this node.

EmitFlags **EMIT_FLAG_COLOR** = `8`

If enabled, the particle starts with the color defined by this node.

EmitFlags **EMIT_FLAG_CUSTOM** = `16`

If enabled, the particle starts with the `CUSTOM` data defined by this node.

Property Descriptions

EmitFlags **flags** = `31` 🔗

- void **set_flags**(value: EmitFlags)
- EmitFlags **get_flags**()

Flags used to override the properties defined in the sub-emitter's process material.

VisualShaderNodeParticleEmitter

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Inherited By: VisualShaderNodeParticleBoxEmitter, VisualShaderNodeParticleMeshEmitter, VisualShaderNodeParticleRingEmitter, VisualShaderNodeParticleSphereEmitter

A base class for particle emitters.

Description

Particle emitter nodes can be used in "start" step of particle shaders and they define the starting position of the particles. Connect them to the Position output port.

Properties

bool	mode_2d	`false`

Property Descriptions

bool **mode_2d** = `false`

- void **set_mode_2d**(value: bool)

- bool **is_mode_2d**()

If `true`, the result of this emitter is projected to 2D space. By default it is `false` and meant for use in 3D space.

VisualShaderNodeParticleMeshEmitter

Inherits: VisualShaderNodeParticleEmitter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node that makes particles emitted in a shape defined by a Mesh.

Description

VisualShaderNodeParticleEmitter that makes the particles emitted in a shape of the assigned mesh. It will emit from the mesh's surfaces, either all or only the specified one.

Properties

Mesh	mesh	
int	surface_index	`0`
bool	use_all_surfaces	`true`

Property Descriptions

Mesh **mesh**

- void **set_mesh**(value: Mesh)
- Mesh **get_mesh**()

The Mesh that defines emission shape.

---

int **surface_index** = `0`

- void **set_surface_index**(value: int)
- int **get_surface_index**()

Index of the surface that emits particles. use_all_surfaces must be `false` for this to take effect.

---

bool **use_all_surfaces** = `true`

- void **set_use_all_surfaces**(value: bool)
- bool **is_use_all_surfaces**()

If `true`, the particles will emit from all surfaces of the mesh.

VisualShaderNodeParticleMultiplyByAxisAngle

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader helper node for multiplying position and rotation of particles.

Description

This node helps to multiply a position input vector by rotation using specific axis. Intended to work with emitters.

Properties

bool	degrees_mode	`true`

Property Descriptions

bool **degrees_mode** = `true` 🔗

- void **set_degrees_mode**(value: bool)
- bool **is_degrees_mode**()

If `true`, the angle will be interpreted in degrees instead of radians.

VisualShaderNodeParticleOutput

Inherits: VisualShaderNodeOutput **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Visual shader node that defines output values for particle emitting.

Description

This node defines how particles are emitted. It allows to customize e.g. position and velocity. Available ports are different depending on which function this node is inside (start, process, collision) and whether custom data is enabled.

VisualShaderNodeParticleRandomness

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Visual shader node for randomizing particle values.

Description

Randomness node will output pseudo-random values of the given type based on the specified minimum and maximum values.

Properties

OpType	op_type	`0`

Enumerations

enum **OpType**: 🔗

OpType **OP_TYPE_SCALAR** = `0`

A floating-point scalar.

OpType **OP_TYPE_VECTOR_2D** = `1`

A 2D vector type.

OpType **OP_TYPE_VECTOR_3D** = `2`

A 3D vector type.

OpType **OP_TYPE_VECTOR_4D** = `3`

A 4D vector type.

OpType **OP_TYPE_MAX** = `4`

Represents the size of the OpType enum.

Property Descriptions

OpType **op_type** = `0` 🔗

- void **set_op_type**(value: OpType)
- OpType **get_op_type**()

A type of operands and returned value.

VisualShaderNodeParticleRingEmitter

Inherits: VisualShaderNodeParticleEmitter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node that makes particles emitted in a ring shape.

Description

VisualShaderNodeParticleEmitter that makes the particles emitted in ring shape with the specified inner and outer radii and height.

VisualShaderNodeParticleSphereEmitter

Inherits: VisualShaderNodeParticleEmitter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node that makes particles emitted in a sphere shape.

Description

VisualShaderNodeParticleEmitter that makes the particles emitted in sphere shape with the specified inner and outer radii.

VisualShaderNodeProximityFade

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node representing proximity fade effect.

Description

The proximity fade effect fades out each pixel based on its distance to another object.

VisualShaderNodeRandomRange

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node that generates a pseudo-random scalar.

Description

Random range node will output a pseudo-random scalar value in the specified range, based on the seed. The value is always the same for the given seed and range, so you should provide a changing input, e.g. by using time.

VisualShaderNodeRemap

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node for remap function.

Description

Remap will transform the input range into output range, e.g. you can change a `0..1` value to `-2..2` etc. See @GlobalScope.remap for more details.

VisualShaderNodeReroute

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A node that allows rerouting a connection within the visual shader graph.

Description

Automatically adapts its port type to the type of the incoming connection and ensures valid connections.

Methods

PortType	get_port_type() const

Method Descriptions

PortType **get_port_type**() const 🔗

Returns the port type of the reroute node.

VisualShaderNodeResizableBase

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Inherited By: VisualShaderNodeCurveTexture, VisualShaderNodeCurveXYZTexture, VisualShaderNodeFrame, VisualShaderNodeGroupBase

Base class for resizable nodes in a visual shader graph.

Description

Resizable nodes have a handle that allows the user to adjust their size as needed.

Properties

Vector2	size	`Vector2(0, 0)`

Property Descriptions

Vector2 **size** = `Vector2(0, 0)` 🔗

- void **set_size**(value: Vector2)
- Vector2 **get_size**()

The size of the node in the visual shader graph.

VisualShaderNodeRotationByAxis

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node that modifies the rotation of the object using a rotation matrix.

Description

RotationByAxis node will transform the vertices of a mesh with specified axis and angle in radians. It can be used to rotate an object in an arbitrary axis.

VisualShaderNodeSample3D

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Inherited By: VisualShaderNodeTexture2DArray, VisualShaderNodeTexture3D

A base node for nodes which samples 3D textures in the visual shader graph.

Description

A virtual class, use the descendants instead.

Properties

Source	source	`0`

Enumerations

enum **Source**:

Source **SOURCE_TEXTURE** = `0`

Creates internal uniform and provides a way to assign it within node.

Source **SOURCE_PORT** = `1`

Use the uniform texture from sampler port.

Source **SOURCE_MAX** = `2`

Represents the size of the Source enum.

Property Descriptions

Source **source** = `0` 🔗

- void **set_source**(value: Source)
- Source **get_source**()

An input source type.

VisualShaderNodeScreenNormalWorldSpace

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node that unpacks the screen normal texture in World Space.

Description

The ScreenNormalWorldSpace node allows to create outline effects.

VisualShaderNodeScreenUVToSDF

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A function to convert screen UV to an SDF (signed-distance field), to be used within the visual shader graph.

Description

Translates to `screen_uv_to_sdf(uv)` in the shader language. If the UV port isn't connected, `SCREEN_UV` is used instead.

VisualShaderNodeSDFRaymarch

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

SDF raymarching algorithm to be used within the visual shader graph.

Description

Casts a ray against the screen SDF (signed-distance field) and returns the distance travelled.

VisualShaderNodeSDFToScreenUV

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A function to convert an SDF (signed-distance field) to screen UV, to be used within the visual shader graph.

Description

Translates to `sdf_to_screen_uv(sdf_pos)` in the shader language.

VisualShaderNodeSmoothStep

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Calculates a SmoothStep function within the visual shader graph.

Description

Translates to `smoothstep(edge0, edge1, x)` in the shader language.

Returns `0.0` if `x` is smaller than `edge0` and `1.0` if `x` is larger than `edge1`. Otherwise, the return value is interpolated between `0.0` and `1.0` using Hermite polynomials.

Properties

OpType	op_type	`0`

Enumerations

enum **OpType**: 🔗

OpType **OP_TYPE_SCALAR** = `0`

A floating-point scalar type.

OpType **OP_TYPE_VECTOR_2D** = `1`

A 2D vector type.

OpType **OP_TYPE_VECTOR_2D_SCALAR** = `2`

The `x` port uses a 2D vector type. The first two ports use a floating-point scalar type.

OpType **OP_TYPE_VECTOR_3D** = `3`

A 3D vector type.

OpType **OP_TYPE_VECTOR_3D_SCALAR** = `4`

The `x` port uses a 3D vector type. The first two ports use a floating-point scalar type.

OpType **OP_TYPE_VECTOR_4D** = `5`

A 4D vector type.

OpType **OP_TYPE_VECTOR_4D_SCALAR** = `6`

The `a` and `b` ports use a 4D vector type. The `weight` port uses a scalar type.

OpType **OP_TYPE_MAX** = `7`

Represents the size of the OpType enum.

Property Descriptions

OpType **op_type** = `0` 🔗

- void **set_op_type**(value: OpType)

- OpType **get_op_type**()

A type of operands and returned value.

VisualShaderNodeStep

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Calculates a Step function within the visual shader graph.

Description

Translates to `step(edge, x)` in the shader language.

Returns `0.0` if `x` is smaller than `edge` and `1.0` otherwise.

Properties

OpType	op_type	`0`

Enumerations

enum **OpType**: 🔗

OpType **OP_TYPE_SCALAR** = `0`

A floating-point scalar type.

OpType **OP_TYPE_VECTOR_2D** = `1`

A 2D vector type.

OpType **OP_TYPE_VECTOR_2D_SCALAR** = `2`

The `x` port uses a 2D vector type, while the `edge` port uses a floating-point scalar type.

OpType **OP_TYPE_VECTOR_3D** = `3`

A 3D vector type.

OpType **OP_TYPE_VECTOR_3D_SCALAR** = `4`

The `x` port uses a 3D vector type, while the `edge` port uses a floating-point scalar type.

OpType **OP_TYPE_VECTOR_4D** = `5`

A 4D vector type.

OpType **OP_TYPE_VECTOR_4D_SCALAR** = `6`

The `a` and `b` ports use a 4D vector type. The `weight` port uses a scalar type.

OpType **OP_TYPE_MAX** = `7`

Represents the size of the OpType enum.

Property Descriptions

OpType **op_type** = `0` 🔗

- void **set_op_type**(value: OpType)
- OpType **get_op_type**()

A type of operands and returned value.

VisualShaderNodeSwitch

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A selector function for use within the visual shader graph.

Description

Returns an associated value of the op_type type if the provided boolean value is `true` or `false`.

Properties

OpType	op_type	`0`

Enumerations

enum **OpType**:

OpType **OP_TYPE_FLOAT** = `0`

A floating-point scalar.

OpType **OP_TYPE_INT** = `1`

An integer scalar.

OpType **OP_TYPE_UINT** = `2`

An unsigned integer scalar.

OpType **OP_TYPE_VECTOR_2D** = `3`

A 2D vector type.

OpType **OP_TYPE_VECTOR_3D** = `4`

A 3D vector type.

OpType **OP_TYPE_VECTOR_4D** = `5`

A 4D vector type.

OpType **OP_TYPE_BOOLEAN** = `6`

A boolean type.

OpType **OP_TYPE_TRANSFORM** = `7`

A transform type.

OpType **OP_TYPE_MAX** = `8`

Represents the size of the OpType enum.

Property Descriptions

OpType **op_type** = `0` 🔗

- void **set_op_type**(value: OpType)
- OpType **get_op_type**()

A type of operands and returned value.

VisualShaderNodeTexture

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Performs a 2D texture lookup within the visual shader graph.

Description

Performs a lookup operation on the provided texture, with support for multiple texture sources to choose from.

Properties

Source	source	0
Texture2D	texture	
TextureType	texture_type	0

Enumerations

enum **Source**: 🔗

Source **SOURCE_TEXTURE** = 0

Use the texture given as an argument for this function.

Source **SOURCE_SCREEN** = `1`

Use the current viewport's texture as the source.

Source **SOURCE_2D_TEXTURE** = `2`

Use the texture from this shader's texture built-in (e.g. a texture of a Sprite2D).

Source **SOURCE_2D_NORMAL** = `3`

Use the texture from this shader's normal map built-in.

Source **SOURCE_DEPTH** = `4`

Use the depth texture captured during the depth prepass. Only available when the depth prepass is used (i.e. in spatial shaders and in the forward_plus or gl_compatibility renderers).

Source **SOURCE_PORT** = `5`

Use the texture provided in the input port for this function.

Source **SOURCE_3D_NORMAL** = `6`

Use the normal buffer captured during the depth prepass. Only available when the normal-roughness buffer is available (i.e. in spatial shaders and in the forward_plus renderer).

Source **SOURCE_ROUGHNESS** = `7`

Use the roughness buffer captured during the depth prepass. Only available when the normal-roughness buffer is available (i.e. in spatial shaders and in the forward_plus renderer).

Source **SOURCE_MAX** = `8`

Represents the size of the Source enum.

---

enum **TextureType**:

TextureType **TYPE_DATA** = `0`

No hints are added to the uniform declaration.

TextureType **TYPE_COLOR** = `1`

Adds `source_color` as hint to the uniform declaration for proper sRGB to linear conversion.

TextureType **TYPE_NORMAL_MAP** = `2`

Adds `hint_normal` as hint to the uniform declaration, which internally converts the texture for proper usage as normal map.

TextureType **TYPE_MAX** = `3`

Represents the size of the TextureType enum.

Property Descriptions

Source **source** = `0`

- void **set_source**(value: Source)
- Source **get_source**()

Determines the source for the lookup. See Source for options.

Texture2D **texture**

- void **set_texture**(value: Texture2D)
- Texture2D **get_texture**()

The source texture, if needed for the selected source.

TextureType **texture_type** = 0

- void **set_texture_type**(value: TextureType)
- TextureType **get_texture_type**()

Specifies the type of the texture if source is set to SOURCE_TEXTURE. See TextureType for options.

VisualShaderNodeTexture2DArray

Inherits: VisualShaderNodeSample3D **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A 2D texture uniform array to be used within the visual shader graph.

Description

Translated to `uniform sampler2DArray` in the shader language.

Properties

Texture2DArray	texture_array

Property Descriptions

Texture2DArray **texture_array** 🔗

- void **set_texture_array**(value: Texture2DArray)
- Texture2DArray **get_texture_array**()

A source texture array. Used if VisualShaderNodeSample3D.source is set to VisualShaderNodeSample3D.SOURCE_TEXTURE.

VisualShaderNodeTexture2DArrayParameter

Inherits: VisualShaderNodeTextureParameter **<** VisualShaderNodeParameter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node for shader parameter (uniform) of type Texture2DArray.

Description

This parameter allows to provide a collection of textures for the shader. You can use VisualShaderNodeTexture2DArray to extract the textures from array.

VisualShaderNodeTexture2DParameter

Inherits: VisualShaderNodeTextureParameter **<** VisualShaderNodeParameter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Provides a 2D texture parameter within the visual shader graph.

Description

Translated to `uniform sampler2D` in the shader language.

VisualShaderNodeTexture3D

Inherits: VisualShaderNodeSample3D **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Performs a 3D texture lookup within the visual shader graph.

Description

Performs a lookup operation on the provided texture, with support for multiple texture sources to choose from.

Properties

Texture3D	texture

Property Descriptions

Texture3D **texture** 🔗

- void **set_texture**(value: Texture3D)
- Texture3D **get_texture**()

A source texture. Used if VisualShaderNodeSample3D.source is set to VisualShaderNodeSample3D.SOURCE_TEXTURE.

VisualShaderNodeTexture3DParameter

Inherits: VisualShaderNodeTextureParameter **<** VisualShaderNodeParameter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Provides a 3D texture parameter within the visual shader graph.

Description

Translated to `uniform sampler3D` in the shader language.

VisualShaderNodeTextureParameter

Inherits: VisualShaderNodeParameter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Inherited By: VisualShaderNodeCubemapParameter, VisualShaderNodeTexture2DArrayParameter, VisualShaderNodeTexture2DParameter, VisualShaderNodeTexture3DParameter, VisualShaderNodeTextureParameterTriplanar

Performs a uniform texture lookup within the visual shader graph.

Description

Performs a lookup operation on the texture provided as a uniform for the shader.

Properties

ColorDefault	color_default	0
TextureFilter	texture_filter	0
TextureRepeat	texture_repeat	0

TextureSource	texture_source	`0`
TextureType	texture_type	`0`

Enumerations

enum **TextureType**: 🔗

TextureType **TYPE_DATA** = `0`

No hints are added to the uniform declaration.

TextureType **TYPE_COLOR** = `1`

Adds `source_color` as hint to the uniform declaration for proper sRGB to linear conversion.

TextureType **TYPE_NORMAL_MAP** = `2`

Adds `hint_normal` as hint to the uniform declaration, which internally converts the texture for proper usage as normal map.

TextureType **TYPE_ANISOTROPY** = `3`

Adds `hint_anisotropy` as hint to the uniform declaration to use for a flowmap.

TextureType **TYPE_MAX** = `4`

Represents the size of the TextureType enum.

enum **ColorDefault**: 🔗

ColorDefault **COLOR_DEFAULT_WHITE** = `0`

Defaults to fully opaque white color.

ColorDefault **COLOR_DEFAULT_BLACK** = `1`

Defaults to fully opaque black color.

ColorDefault **COLOR_DEFAULT_TRANSPARENT** = `2`

Defaults to fully transparent black color.

ColorDefault **COLOR_DEFAULT_MAX** = `3`

Represents the size of the ColorDefault enum.

---

enum **TextureFilter**: 🔗

TextureFilter **FILTER_DEFAULT** = `0`

Sample the texture using the filter determined by the node this shader is attached to.

TextureFilter **FILTER_NEAREST** = `1`

The texture filter reads from the nearest pixel only. This makes the texture look pixelated from up close, and grainy from a distance (due to mipmaps not being sampled).

TextureFilter **FILTER_LINEAR** = `2`

The texture filter blends between the nearest 4 pixels. This makes the texture look smooth from up close, and grainy from a distance (due to mipmaps not being sampled).

TextureFilter **FILTER_NEAREST_MIPMAP** = `3`

The texture filter reads from the nearest pixel and blends between the nearest 2 mipmaps (or uses the nearest mipmap if ProjectSettings.rendering/textures/default_filters/use_nearest_mipmap_filter is `true`). This makes the texture look pixelated from up close, and smooth from a distance.

Use this for non-pixel art textures that may be viewed at a low scale (e.g. due to Camera2D zoom or sprite scaling), as mipmaps are important to smooth out pixels that are smaller than on-screen pixels.

TextureFilter **FILTER_LINEAR_MIPMAP** = `4`

The texture filter blends between the nearest 4 pixels and between the nearest 2 mipmaps (or uses the nearest mipmap if ProjectSettings.rendering/textures/default_filters/use_nearest_mipmap_filter is `true`). This makes the texture look smooth from up close, and smooth from a distance.

Use this for non-pixel art textures that may be viewed at a low scale (e.g. due to Camera2D zoom or sprite scaling), as mipmaps are important to smooth out pixels that are smaller than on-screen pixels.

TextureFilter **FILTER_NEAREST_MIPMAP_ANISOTROPIC** = `5`

The texture filter reads from the nearest pixel and blends between 2 mipmaps (or uses the nearest mipmap if ProjectSettings.rendering/textures/default_filters/use_nearest_mipmap_filter is `true`) based on the angle between the surface and the camera view. This makes the texture look pixelated from up close, and smooth from a distance. Anisotropic filtering improves texture quality on surfaces that are almost in line with the camera, but is slightly slower. The anisotropic filtering level can be changed by adjusting

ProjectSettings.rendering/textures/default_filters/anisotropic_filtering_level.

Note: This texture filter is rarely useful in 2D projects. FILTER_NEAREST_MIPMAP is usually more appropriate in this case.

TextureFilter **FILTER_LINEAR_MIPMAP_ANISOTROPIC** = `6`

The texture filter blends between the nearest 4 pixels and blends between 2 mipmaps (or uses the nearest mipmap if ProjectSettings.rendering/textures/default_filters/use_nearest_mipmap_filter is `true`) based on the angle between the surface and the camera view. This makes the texture look smooth from up close, and smooth from a distance. Anisotropic filtering improves texture quality on surfaces that are almost in line with the camera, but is slightly slower. The anisotropic filtering level can be changed by adjusting ProjectSettings.rendering/textures/default_filters/anisotropic_filtering_level.

Note: This texture filter is rarely useful in 2D projects. FILTER_LINEAR_MIPMAP is usually more appropriate in this case.

TextureFilter **FILTER_MAX** = `7`

Represents the size of the TextureFilter enum.

---

enum **TextureRepeat**: 🔗

TextureRepeat **REPEAT_DEFAULT** = `0`

Sample the texture using the repeat mode determined by the node this shader is attached to.

TextureRepeat **REPEAT_ENABLED** = `1`

Texture will repeat normally.

TextureRepeat **REPEAT_DISABLED** = `2`

Texture will not repeat.

TextureRepeat **REPEAT_MAX** = `3`

Represents the size of the TextureRepeat enum.

---

enum **TextureSource**: 🔗

TextureSource **SOURCE_NONE** = `0`

The texture source is not specified in the shader.

TextureSource **SOURCE_SCREEN** = `1`

The texture source is the screen texture which captures all opaque objects drawn this frame.

TextureSource **SOURCE_DEPTH** = `2`

The texture source is the depth texture from the depth prepass.

TextureSource **SOURCE_NORMAL_ROUGHNESS** = `3`

The texture source is the normal-roughness buffer from the depth prepass.

TextureSource **SOURCE_MAX** = `4`

Represents the size of the TextureSource enum.

Property Descriptions

ColorDefault **color_default** = `0` 🔗

- void **set_color_default**(value: ColorDefault)
- ColorDefault **get_color_default**()

Sets the default color if no texture is assigned to the uniform.

TextureFilter **texture_filter** = 0

- void **set_texture_filter**(value: TextureFilter)
- TextureFilter **get_texture_filter**()

Sets the texture filtering mode. See TextureFilter for options.

TextureRepeat **texture_repeat** = 0

- void **set_texture_repeat**(value: TextureRepeat)
- TextureRepeat **get_texture_repeat**()

Sets the texture repeating mode. See TextureRepeat for options.

TextureSource **texture_source** = 0

- void **set_texture_source**(value: TextureSource)
- TextureSource **get_texture_source**()

Sets the texture source mode. Used for reading from the screen, depth, or normal_roughness texture. See TextureSource for options.

TextureType **texture_type** = 0

- void **set_texture_type**(value: TextureType)
- TextureType **get_texture_type**()

Defines the type of data provided by the source texture. See TextureType for options.

VisualShaderNodeTextureParameterTriplanar

Inherits: VisualShaderNodeTextureParameter **<** VisualShaderNodeParameter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Performs a uniform texture lookup with triplanar within the visual shader graph.

Description

Performs a lookup operation on the texture provided as a uniform for the shader, with support for triplanar mapping.

VisualShaderNodeTextureSDF

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Performs an SDF (signed-distance field) texture lookup within the visual shader graph.

Description

Translates to `texture_sdf(sdf_pos)` in the shader language.

VisualShaderNodeTextureSDFNormal

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Performs an SDF (signed-distance field) normal texture lookup within the visual shader graph.

Description

Translates to `texture_sdf_normal(sdf_pos)` in the shader language.

VisualShaderNodeTransformCompose

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Composes a Transform3D from four Vector3s within the visual shader graph.

Description

Creates a 4×4 transform matrix using four vectors of type `vec3`. Each vector is one row in the matrix and the last column is a `vec4(0, 0, 0, 1)`.

VisualShaderNodeTransformConstant

Inherits: VisualShaderNodeConstant **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A Transform3D constant for use within the visual shader graph.

Description

A constant Transform3D, which can be used as an input node.

Properties

Transform3D	constant	`Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)`

Property Descriptions

Transform3D **constant** = `Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)` 🔗

- void **set_constant**(value: Transform3D)
- Transform3D **get_constant**()

A Transform3D constant which represents the state of this node.

VisualShaderNodeTransformDecompose

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Decomposes a Transform3D into four Vector3s within the visual shader graph.

Description

Takes a 4×4 transform matrix and decomposes it into four `vec3` values, one from each row of the matrix.

VisualShaderNodeTransformFunc

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Computes a Transform3D function within the visual shader graph.

Description

Computes an inverse or transpose function on the provided Transform3D.

Properties

Function	function	`0`

Enumerations

enum **Function**: 🔗

Function **FUNC_INVERSE** = `0`

Perform the inverse operation on the Transform3D matrix.

Function **FUNC_TRANSPOSE** = `1`

Perform the transpose operation on the Transform3D matrix.

Function **FUNC_MAX** = `2`

Represents the size of the Function enum.

Property Descriptions

Function **function** = `0` 🔗

- void **set_function**(value: Function)
- Function **get_function**()

The function to be computed. See Function for options.

VisualShaderNodeTransformOp

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A Transform3D operator to be used within the visual shader graph.

Description

Applies operator to two transform (4×4 matrices) inputs.

Properties

Operator	operator	`0`

Enumerations

enum **Operator**: 🔗

Operator **OP_AxB** = `0`

Multiplies transform `a` by the transform `b`.

Operator **OP_BxA** = `1`

Multiplies transform `b` by the transform `a`.

Operator **OP_AxB_COMP** = `2`

Performs a component-wise multiplication of transform `a` by the transform `b`.

Operator **OP_BxA_COMP** = `3`

Performs a component-wise multiplication of transform `b` by the transform `a`.

Operator **OP_ADD** = `4`

Adds two transforms.

Operator **OP_A_MINUS_B** = `5`

Subtracts the transform `a` from the transform `b`.

Operator **OP_B_MINUS_A** = `6`

Subtracts the transform `b` from the transform `a`.

Operator **OP_A_DIV_B** = `7`

Divides the transform `a` by the transform `b`.

Operator **OP_B_DIV_A** = `8`

Divides the transform `b` by the transform `a`.

Operator **OP_MAX** = `9`

Represents the size of the Operator enum.

Property Descriptions

Operator **operator** = `0` 🔗

- void **set_operator**(value: Operator)
- Operator **get_operator**()

The type of the operation to be performed on the transforms. See Operator for options.

VisualShaderNodeTransformParameter

Inherits: VisualShaderNodeParameter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A Transform3D parameter for use within the visual shader graph.

Description

Translated to `uniform mat4` in the shader language.

Properties

Transform3D	default_value	`Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)`
bool	default_value_enabled	`false`

Property Descriptions

Transform3D **default_value** = `Transform3D(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)` 🔗

- void **set_default_value**(value: Transform3D)
- Transform3D **get_default_value**()

A default value to be assigned within the shader.

---

bool **default_value_enabled** = `false`

- void **set_default_value_enabled**(value: bool)
- bool **is_default_value_enabled**()

Enables usage of the default_value.

VisualShaderNodeTransformVecMult

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Multiplies a Transform3D and a Vector3 within the visual shader graph.

Description

A multiplication operation on a transform (4×4 matrix) and a vector, with support for different multiplication operators.

Properties

Operator	operator	`0`

Enumerations

enum **Operator**: 🔗

Operator **OP_AxB** = `0`

Multiplies transform `a` by the vector `b`.

Operator **OP_BxA** = `1`

Multiplies vector `b` by the transform `a`.

Operator **OP_3x3_AxB** = `2`

Multiplies transform `a` by the vector `b`, skipping the last row and column of the transform.

Operator **OP_3x3_BxA** = `3`

Multiplies vector `b` by the transform `a`, skipping the last row and column of the transform.

Operator **OP_MAX** = `4`

Represents the size of the Operator enum.

Property Descriptions

Operator **operator** = `0` 🔗

- void **set_operator**(value: Operator)
- Operator **get_operator**()

The multiplication type to be performed. See Operator for options.

VisualShaderNodeUIntConstant

Inherits: VisualShaderNodeConstant **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

An unsigned scalar integer constant to be used within the visual shader graph.

Description

Translated to `uint` in the shader language.

Properties

int	constant	`0`

Property Descriptions

int **constant** = `0` 🔗

- void **set_constant**(value: int)
- int **get_constant**()

An unsigned integer constant which represents a state of this node.

VisualShaderNodeUIntFunc

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

An unsigned scalar integer function to be used within the visual shader graph.

Description

Accept an unsigned integer scalar (`x`) to the input port and transform it according to function.

Properties

Function	function	`0`

Enumerations

enum **Function**: 🔗

Function **FUNC_NEGATE** = `0`

Negates the `x` using `-(x)`.

Function **FUNC_BITWISE_NOT** = `1`

Returns the result of bitwise `NOT` operation on the integer. Translates to `~a` in the Godot Shader Language.

Function **FUNC_MAX** = `2`

Represents the size of the Function enum.

Property Descriptions

Function **function** = `0` 🔗

- void **set_function**(value: Function)
- Function **get_function**()

A function to be applied to the scalar. See Function for options.

VisualShaderNodeUIntOp

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

An unsigned integer scalar operator to be used within the visual shader graph.

Description

Applies operator to two unsigned integer inputs: `a` and `b`.

Properties

Operator	operator	`0`

Enumerations

enum **Operator**: 🔗

Operator **OP_ADD** = `0`

Sums two numbers using `a + b`.

Operator **OP_SUB** = `1`

Subtracts two numbers using `a - b`.

Operator **OP_MUL** = `2`

Multiplies two numbers using `a * b`.

Operator **OP_DIV** = `3`

Divides two numbers using `a / b`.

Operator **OP_MOD** = `4`

Calculates the remainder of two numbers using `a % b`.

Operator **OP_MAX** = `5`

Returns the greater of two numbers. Translates to `max(a, b)` in the Godot Shader Language.

Operator **OP_MIN** = `6`

Returns the lesser of two numbers. Translates to `max(a, b)` in the Godot Shader Language.

Operator **OP_BITWISE_AND** = `7`

Returns the result of bitwise `AND` operation on the integer. Translates to `a & b` in the Godot Shader Language.

Operator **OP_BITWISE_OR** = `8`

Returns the result of bitwise `OR` operation for two integers. Translates to `a | b` in the Godot Shader Language.

Operator **OP_BITWISE_XOR** = `9`

Returns the result of bitwise `XOR` operation for two integers. Translates to `a ^ b` in the Godot Shader Language.

Operator **OP_BITWISE_LEFT_SHIFT** = `10`

Returns the result of bitwise left shift operation on the integer. Translates to `a << b` in the Godot Shader Language.

Operator **OP_BITWISE_RIGHT_SHIFT** = `11`

Returns the result of bitwise right shift operation on the integer. Translates to `a >> b` in the Godot Shader Language.

Operator **OP_ENUM_SIZE** = `12`

Represents the size of the Operator enum.

Property Descriptions

Operator **operator** = `0`

- void **set_operator**(value: Operator)
- Operator **get_operator**()

An operator to be applied to the inputs. See Operator for options.

VisualShaderNodeUIntParameter

Inherits: VisualShaderNodeParameter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node for shader parameter (uniform) of type unsigned int.

Description

A VisualShaderNodeParameter of type unsigned int. Offers additional customization for range of accepted values.

Properties

int	default_value	`0`
bool	default_value_enabled	`false`

Property Descriptions

int **default_value** = `0`

- void **set_default_value**(value: int)
- int **get_default_value**()

Default value of this parameter, which will be used if not set externally. default_value_enabled must be enabled; defaults to `0` otherwise.

---

bool **default_value_enabled** = `false`

- void **set_default_value_enabled**(value: bool)
- bool **is_default_value_enabled**()

If `true`, the node will have a custom default value.

VisualShaderNodeUVFunc

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Contains functions to modify texture coordinates (`uv`) to be used within the visual shader graph.

Description

UV functions are similar to Vector2 functions, but the input port of this node uses the shader's UV value by default.

Properties

Function	function	`0`

Enumerations

enum **Function**: 🔗

Function **FUNC_PANNING** = `0`

Translates `uv` by using `scale` and `offset` values using the following formula: `uv = uv + offset * scale`. `uv` port is connected to `UV` built-in by default.

Function **FUNC_SCALING** = `1`

Scales `uv` by using `scale` and `pivot` values using the following formula: `uv = (uv - pivot) * scale + pivot`. `uv` port is connected to `UV` built-in by default.

Function **FUNC_MAX** = `2`

Represents the size of the Function enum.

Property Descriptions

Function **function** = `0` 🔗

- void **set_function**(value: Function)
- Function **get_function**()

A function to be applied to the texture coordinates. See Function for options.

VisualShaderNodeUVPolarCoord

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node that modifies the texture UV using polar coordinates.

Description

UV polar coord node will transform UV values into polar coordinates, with specified scale, zoom strength and repeat parameters. It can be used to create various swirl distortions.

VisualShaderNodeVarying

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Inherited By: VisualShaderNodeVaryingGetter, VisualShaderNodeVaryingSetter

A visual shader node that represents a "varying" shader value.

Description

Varying values are shader variables that can be passed between shader functions, e.g. from Vertex shader to Fragment shader.

Properties

String	varying_name	`"[None]"`
VaryingType	varying_type	`0`

Property Descriptions

String **varying_name** = `"[None]"` 🔗

- void **set_varying_name**(value: String)
- String **get_varying_name**()

Name of the variable. Must be unique.

VaryingType **varying_type** = 0 🔗

- void **set_varying_type**(value: VaryingType)
- VaryingType **get_varying_type**()

Type of the variable. Determines where the variable can be accessed.

VisualShaderNodeVaryingGetter

Inherits: VisualShaderNodeVarying **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node that gets a value of a varying.

Description

Outputs a value of a varying defined in the shader. You need to first create a varying that can be used in the given function, e.g. varying getter in Fragment shader requires a varying with mode set to VisualShader.VARYING_MODE_VERTEX_TO_FRAG_LIGHT.

VisualShaderNodeVaryingSetter

Inherits: VisualShaderNodeVarying **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node that sets a value of a varying.

Description

Inputs a value to a varying defined in the shader. You need to first create a varying that can be used in the given function, e.g. varying setter in Fragment shader requires a varying with mode set to VisualShader.VARYING_MODE_FRAG_TO_LIGHT.

VisualShaderNodeVec2Constant

Inherits: VisualShaderNodeConstant **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A Vector2 constant to be used within the visual shader graph.

Description

A constant Vector2, which can be used as an input node.

Properties

Vector2	constant	`Vector2(0, 0)`

Property Descriptions

Vector2 **constant** = `Vector2(0, 0)`

- void **set_constant**(value: Vector2)
- Vector2 **get_constant**()

A Vector2 constant which represents the state of this node.

VisualShaderNodeVec2Parameter

Inherits: VisualShaderNodeParameter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A Vector2 parameter to be used within the visual shader graph.

Description

Translated to `uniform vec2` in the shader language.

Properties

Vector2	default_value	`Vector2(0, 0)`
bool	default_value_enabled	`false`

Property Descriptions

Vector2 **default_value** = `Vector2(0, 0)` 🔗

- void **set_default_value**(value: Vector2)
- Vector2 **get_default_value**()

A default value to be assigned within the shader.

---

bool **default_value_enabled** = `false` 🔗

- void **set_default_value_enabled**(value: bool)
- bool **is_default_value_enabled**()

Enables usage of the default_value.

VisualShaderNodeVec3Constant

Inherits: VisualShaderNodeConstant **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A Vector3 constant to be used within the visual shader graph.

Description

A constant Vector3, which can be used as an input node.

Properties

Vector3	constant	`Vector3(0, 0, 0)`

Property Descriptions

Vector3 **constant** = `Vector3(0, 0, 0)` 🔗

- void **set_constant**(value: Vector3)
- Vector3 **get_constant**()

A Vector3 constant which represents the state of this node.

VisualShaderNodeVec3Parameter

Inherits: VisualShaderNodeParameter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A Vector3 parameter to be used within the visual shader graph.

Description

Translated to `uniform vec3` in the shader language.

Properties

Vector3	default_value	`Vector3(0, 0, 0)`
bool	default_value_enabled	`false`

Property Descriptions

Vector3 **default_value** = `Vector3(0, 0, 0)` 🔗

- void **set_default_value**(value: Vector3)
- Vector3 **get_default_value**()

A default value to be assigned within the shader.

---

bool **default_value_enabled** = `false` 🔗

- void **set_default_value_enabled**(value: bool)
- bool **is_default_value_enabled**()

Enables usage of the default_value.

VisualShaderNodeVec4Constant

Inherits: VisualShaderNodeConstant **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A 4D vector constant to be used within the visual shader graph.

Description

A constant 4D vector, which can be used as an input node.

Properties

Quaternion	constant	`Quaternion(0, 0, 0, 1)`

Property Descriptions

Quaternion **constant** = `Quaternion(0, 0, 0, 1)` 🔗

- void **set_constant**(value: Quaternion)
- Quaternion **get_constant**()

A 4D vector (represented as a Quaternion) constant which represents the state of this node.

VisualShaderNodeVec4Parameter

Inherits: VisualShaderNodeParameter **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A 4D vector parameter to be used within the visual shader graph.

Description

Translated to `uniform vec4` in the shader language.

Properties

Vector4	default_value	`Vector4(0, 0, 0, 0)`
bool	default_value_enabled	`false`

Property Descriptions

Vector4 **default_value** = `Vector4(0, 0, 0, 0)` 🔗

- void **set_default_value**(value: Vector4)
- Vector4 **get_default_value**()

A default value to be assigned within the shader.

---

bool **default_value_enabled** = `false`

- void **set_default_value_enabled**(value: bool)
- bool **is_default_value_enabled**()

Enables usage of the default_value.

VisualShaderNodeVectorBase

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

Inherited By: VisualShaderNodeFaceForward, VisualShaderNodeVectorCompose, VisualShaderNodeVectorDecompose, VisualShaderNodeVectorDistance, VisualShaderNodeVectorFunc, VisualShaderNodeVectorLen, VisualShaderNodeVectorOp, VisualShaderNodeVectorRefract

A base type for the nodes that perform vector operations within the visual shader graph.

Description

This is an abstract class. See the derived types for descriptions of the possible operations.

Properties

OpType	op_type	`1`

Enumerations

enum **OpType**: 🔗

OpType **OP_TYPE_VECTOR_2D** = `0`

A 2D vector type.

OpType **OP_TYPE_VECTOR_3D** = `1`

A 3D vector type.

OpType **OP_TYPE_VECTOR_4D** = `2`

A 4D vector type.

OpType **OP_TYPE_MAX** = `3`

Represents the size of the OpType enum.

Property Descriptions

OpType **op_type** = `1` 🔗

- void **set_op_type**(value: OpType)
- OpType **get_op_type**()

A vector type that this operation is performed on.

VisualShaderNodeVectorCompose

Inherits: VisualShaderNodeVectorBase **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Composes a Vector2, Vector3 or 4D vector (represented as a Quaternion) from scalars within the visual shader graph.

Description

Creates a `vec2`, `vec3` or `vec4` using scalar values that can be provided from separate inputs.

VisualShaderNodeVectorDecompose

Inherits: VisualShaderNodeVectorBase **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Decomposes a Vector2, Vector3 or 4D vector (represented as a Quaternion) into scalars within the visual shader graph.

Description

Takes a `vec2`, `vec3` or `vec4` and decomposes it into scalar values that can be used as separate outputs.

VisualShaderNodeVectorDistance

Inherits: VisualShaderNodeVectorBase **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Returns the distance between two points. To be used within the visual shader graph.

Description

Calculates distance from point represented by vector `p0` to vector `p1`.

Translated to `distance(p0, p1)` in the shader language.

VisualShaderNodeVectorFunc

Inherits: VisualShaderNodeVectorBase **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A vector function to be used within the visual shader graph.

Description

A visual shader node able to perform different functions using vectors.

Properties

Function	function	`0`

Enumerations

enum **Function**: 🔗

Function **FUNC_NORMALIZE** = `0`

Normalizes the vector so that it has a length of `1` but points in the same direction.

Function **FUNC_SATURATE** = `1`

Clamps the value between `0.0` and `1.0`.

Function **FUNC_NEGATE** = `2`

Returns the opposite value of the parameter.

Function **FUNC_RECIPROCAL** = `3`

Returns `1/vector`.

Function **FUNC_ABS** = `4`

Returns the absolute value of the parameter.

Function **FUNC_ACOS** = `5`

Returns the arc-cosine of the parameter.

Function **FUNC_ACOSH** = `6`

Returns the inverse hyperbolic cosine of the parameter.

Function **FUNC_ASIN** = `7`

Returns the arc-sine of the parameter.

Function **FUNC_ASINH** = `8`

Returns the inverse hyperbolic sine of the parameter.

Function **FUNC_ATAN** = `9`

Returns the arc-tangent of the parameter.

Function **FUNC_ATANH** = `10`

Returns the inverse hyperbolic tangent of the parameter.

Function **FUNC_CEIL** = `11`

Finds the nearest integer that is greater than or equal to the parameter.

Function **FUNC_COS** = `12`

Returns the cosine of the parameter.

Function **FUNC_COSH** = `13`

Returns the hyperbolic cosine of the parameter.

Function **FUNC_DEGREES** = `14`

Converts a quantity in radians to degrees.

Function **FUNC_EXP** = `15`

Base-e Exponential.

Function **FUNC_EXP2** = `16`

Base-2 Exponential.

Function **FUNC_FLOOR** = `17`

Finds the nearest integer less than or equal to the parameter.

Function **FUNC_FRACT** = `18`

Computes the fractional part of the argument.

Function **FUNC_INVERSE_SQRT** = `19`

Returns the inverse of the square root of the parameter.

Function **FUNC_LOG** = `20`

Natural logarithm.

Function **FUNC_LOG2** = `21`

Base-2 logarithm.

Function **FUNC_RADIANS** = `22`

Converts a quantity in degrees to radians.

Function **FUNC_ROUND** = `23`

Finds the nearest integer to the parameter.

Function **FUNC_ROUNDEVEN** = `24`

Finds the nearest even integer to the parameter.

Function **FUNC_SIGN** = `25`

Extracts the sign of the parameter, i.e. returns `-1` if the parameter is negative, `1` if it's positive and `0` otherwise.

Function **FUNC_SIN** = `26`

Returns the sine of the parameter.

Function **FUNC_SINH** = `27`

Returns the hyperbolic sine of the parameter.

Function **FUNC_SQRT** = `28`

Returns the square root of the parameter.

Function **FUNC_TAN** = `29`

Returns the tangent of the parameter.

Function **FUNC_TANH** = `30`

Returns the hyperbolic tangent of the parameter.

Function **FUNC_TRUNC** = `31`

Returns a value equal to the nearest integer to the parameter whose absolute value is not larger than the absolute value of the parameter.

Function **FUNC_ONEMINUS** = `32`

Returns `1.0 - vector`.

Function **FUNC_MAX** = `33`

Represents the size of the Function enum.

Property Descriptions

Function **function** = `0` 🔗

- void **set_function**(value: Function)
- Function **get_function**()

The function to be performed. See Function for options.

VisualShaderNodeVectorLen

Inherits: VisualShaderNodeVectorBase **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Returns the length of a Vector3 within the visual shader graph.

Description

Translated to `length(p0)` in the shader language.

VisualShaderNodeVectorOp

Inherits: VisualShaderNodeVectorBase **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

A vector operator to be used within the visual shader graph.

Description

A visual shader node for use of vector operators. Operates on vector `a` and vector `b`.

Properties

Operator	operator	`0`

Enumerations

enum **Operator**: 🔗

Operator **OP_ADD** = `0`

Adds two vectors.

Operator **OP_SUB** = `1`

Subtracts a vector from a vector.

Operator **OP_MUL** = `2`

Multiplies two vectors.

Operator **OP_DIV** = `3`

Divides vector by vector.

Operator **OP_MOD** = `4`

Returns the remainder of the two vectors.

Operator **OP_POW** = `5`

Returns the value of the first parameter raised to the power of the second, for each component of the vectors.

Operator **OP_MAX** = `6`

Returns the greater of two values, for each component of the vectors.

Operator **OP_MIN** = `7`

Returns the lesser of two values, for each component of the vectors.

Operator **OP_CROSS** = `8`

Calculates the cross product of two vectors.

Operator **OP_ATAN2** = `9`

Returns the arc-tangent of the parameters.

Operator **OP_REFLECT** = `10`

Returns the vector that points in the direction of reflection. `a` is incident vector and `b` is the normal vector.

Operator **OP_STEP** = `11`

Vector step operator. Returns `0.0` if `a` is smaller than `b` and `1.0` otherwise.

Operator **OP_ENUM_SIZE** = `12`

Represents the size of the Operator enum.

Property Descriptions

Operator **operator** = `0`

- void **set_operator**(value: Operator)
- Operator **get_operator**()

The operator to be used. See Operator for options.

VisualShaderNodeVectorRefract

Inherits: VisualShaderNodeVectorBase **<** VisualShaderNode **<** Resource **<** RefCounted **<** Object

Returns the vector that points in the direction of refraction. For use within the visual shader graph.

Description

Translated to `refract(I, N, eta)` in the shader language, where `I` is the incident vector, `N` is the normal vector and `eta` is the ratio of the indices of the refraction.

VisualShaderNodeWorldPositionFromDepth

Inherits: VisualShaderNode **<** Resource **<** RefCounted **<** Object

A visual shader node that calculates the position of the pixel in world space using the depth texture.

Description

The WorldPositionFromDepth node reconstructs the depth position of the pixel in world space. This can be used to obtain world space UVs for projection mapping like Caustics.

VoxelGIData

Inherits: Resource **<** RefCounted **<** Object

Contains baked voxel global illumination data for use in a VoxelGI node.

Description

VoxelGIData contains baked voxel global illumination for use in a VoxelGI node. **VoxelGIData** also offers several properties to adjust the final appearance of the global illumination. These properties can be adjusted at run-time without having to bake the VoxelGI node again.

Note: To prevent text-based scene files (`.tscn`) from growing too much and becoming slow to load and save, always save **VoxelGIData** to an external binary resource file (`.res`) instead of embedding it within the scene. This can be done by clicking the dropdown arrow next to the **VoxelGIData** resource, choosing **Edit**, clicking the floppy disk icon at the top of the Inspector then choosing **Save As...**.

Tutorials

- Third Person Shooter (TPS) Demo [https://godotengine.org/asset-library/asset/2710]

Properties

float	bias	`1.5`
float	dynamic_range	`2.0`
float	energy	`1.0`
bool	interior	`false`
float	normal_bias	`0.0`
float	propagation	`0.5`
bool	use_two_bounces	`true`

Methods

void	allocate(to_cell_xform: Transform3D, aabb: AABB, octree_size: Vector3, octree_cells: PackedByteArray, data_cells: PackedByteArray, distance_field: PackedByteArray, level_counts: PackedInt32Array)
AABB	get_bounds() const
PackedByteArray	get_data_cells() const

PackedInt32Array	get_level_counts() const
PackedByteArray	get_octree_cells() const
Vector3	get_octree_size() const
Transform3D	get_to_cell_xform() const

Property Descriptions

float **bias** = `1.5`

- void **set_bias**(value: float)
- float **get_bias**()

The normal bias to use for indirect lighting and reflections. Higher values reduce self-reflections visible in non-rough materials, at the cost of more visible light leaking and flatter-looking indirect lighting. To prioritize hiding self-reflections over lighting quality, set bias to `0.0` and normal_bias to a value between `1.0` and `2.0`.

float **dynamic_range** = `2.0`

- void **set_dynamic_range**(value: float)
- float **get_dynamic_range**()

The dynamic range to use (`1.0` represents a low dynamic range scene brightness). Higher values can be used to provide brighter indirect lighting, at the cost of more visible color banding in dark areas (both in indirect lighting and reflections). To avoid color banding, it's recommended to use the lowest value that does not result in visible light clipping.

---

float **energy** = `1.0` 🔗

- void **set_energy**(value: float)
- float **get_energy**()

The energy of the indirect lighting and reflections produced by the VoxelGI node. Higher values result in brighter indirect lighting. If indirect lighting looks too flat, try decreasing propagation while increasing energy at the same time. See also use_two_bounces which influences the indirect lighting's effective brightness.

---

bool **interior** = `false` 🔗

- void **set_interior**(value: bool)
- bool **is_interior**()

If `true`, Environment lighting is ignored by the VoxelGI node. If `false`, Environment lighting is taken into account by the VoxelGI node. Environment lighting updates in real-time, which means it can be changed without having to bake the VoxelGI node again.

---

float **normal_bias** = `0.0` 🔗

- void **set_normal_bias**(value: float)
- float **get_normal_bias**()

The normal bias to use for indirect lighting and reflections. Higher values reduce self-reflections visible in non-rough materials, at the cost of more visible light leaking and flatter-looking indirect lighting.

See also bias. To prioritize hiding self-reflections over lighting quality, set bias to `0.0` and normal_bias to a value between `1.0` and `2.0`.

---

float **propagation** = `0.5` 🔗

- void **set_propagation**(value: float)
- float **get_propagation**()

The multiplier to use when light bounces off a surface. Higher values result in brighter indirect lighting. If indirect lighting looks too flat, try decreasing propagation while increasing energy at the same time. See also use_two_bounces which influences the indirect lighting's effective brightness.

---

bool **use_two_bounces** = `true` 🔗

- void **set_use_two_bounces**(value: bool)
- bool **is_using_two_bounces**()

If `true`, performs two bounces of indirect lighting instead of one. This makes indirect lighting look more natural and brighter at a small performance cost. The second bounce is also visible in reflections. If the scene appears too bright after enabling use_two_bounces, adjust propagation and energy.

Method Descriptions

void **allocate**(to_cell_xform: Transform3D, aabb: AABB, octree_size: Vector3, octree_cells: PackedByteArray, data_cells: PackedByteArray, distance_field: PackedByteArray, level_counts: PackedInt32Array) 🔗

There is currently no description for this method. Please help us by contributing one!

AABB **get_bounds**() const

Returns the bounds of the baked voxel data as an AABB, which should match VoxelGI.size after being baked (which only contains the size as a Vector3).

Note: If the size was modified without baking the VoxelGI data, then the value of get_bounds and VoxelGI.size will not match.

PackedByteArray **get_data_cells**() const

There is currently no description for this method. Please help us by contributing one!

PackedInt32Array **get_level_counts**() const

There is currently no description for this method. Please help us by contributing one!

PackedByteArray **get_octree_cells**() const

There is currently no description for this method. Please help us by contributing one!

Vector3 **get_octree_size**() const

There is currently no description for this method. Please help us by contributing one!

Transform3D **get_to_cell_xform**() const

There is currently no description for this method. Please help us by contributing one!

World2D

Inherits: Resource **<** RefCounted **<** Object

A resource that holds all components of a 2D world, such as a canvas and a physics space.

Description

Class that has everything pertaining to a 2D world: A physics space, a canvas, and a sound space. 2D nodes register their resources into the current 2D world.

Tutorials

- Ray-casting

Properties

RID	canvas
PhysicsDirectSpaceState2D	direct_space_state
RID	navigation_map
RID	space

Property Descriptions

RID **canvas** 🔗

- RID **get_canvas**()

The RID of this world's canvas resource. Used by the RenderingServer for 2D drawing.

PhysicsDirectSpaceState2D **direct_space_state** 🔗

- PhysicsDirectSpaceState2D **get_direct_space_state**()

Direct access to the world's physics 2D space state. Used for querying current and potential collisions. When using multi-threaded physics, access is limited to Node._physics_process in the main thread.

RID **navigation_map** 🔗

- RID **get_navigation_map**()

The RID of this world's navigation map. Used by the NavigationServer2D.

RID **space** 🔗

- RID **get_space**()

The RID of this world's physics space resource. Used by the PhysicsServer2D for 2D physics, treating it as both a space and an area.

World3D

Inherits: Resource **<** RefCounted **<** Object

A resource that holds all components of a 3D world, such as a visual scenario and a physics space.

Description

Class that has everything pertaining to a world: A physics space, a visual scenario, and a sound space. 3D nodes register their resources into the current 3D world.

Tutorials

- Ray-casting

Properties

CameraAttributes	camera_attributes
PhysicsDirectSpaceState3D	direct_space_state
Environment	environment
Environment	fallback_environment

RID	navigation_map
RID	scenario
RID	space

Property Descriptions

CameraAttributes **camera_attributes** 🔗

- void **set_camera_attributes**(value: CameraAttributes)
- CameraAttributes **get_camera_attributes**()

The default CameraAttributes resource to use if none set on the Camera3D.

PhysicsDirectSpaceState3D **direct_space_state** 🔗

- PhysicsDirectSpaceState3D **get_direct_space_state**()

Direct access to the world's physics 3D space state. Used for querying current and potential collisions. When using multi-threaded physics, access is limited to Node._physics_process in the main thread.

Environment **environment** 🔗

- void **set_environment**(value: Environment)
- Environment **get_environment**()

The World3D's Environment.

Environment **fallback_environment**

- void **set_fallback_environment**(value: Environment)
- Environment **get_fallback_environment**()

The World3D's fallback environment will be used if environment fails or is missing.

RID **navigation_map**

- RID **get_navigation_map**()

The RID of this world's navigation map. Used by the NavigationServer3D.

RID **scenario**

- RID **get_scenario**()

The World3D's visual scenario.

RID **space**

- RID **get_space**()

The World3D's physics space.

WorldBoundaryShape2D

Inherits: Shape2D **<** Resource **<** RefCounted **<** Object

A 2D world boundary (half-plane) shape used for physics collision.

Description

A 2D world boundary shape, intended for use in physics. **WorldBoundaryShape2D** works like an infinite straight line that forces all physics bodies to stay above it. The line's normal determines which direction is considered as "above" and in the editor, the smaller line over it represents this direction. It can for example be used for endless flat floors.

Properties

float	distance	`0.0`
Vector2	normal	`Vector2(0, -1)`

Property Descriptions

float **distance** = `0.0`

- void **set_distance**(value: float)
- float **get_distance**()

The distance from the origin to the line, expressed in terms of normal (according to its direction and magnitude). Actual absolute distance from the origin to the line can be calculated as `abs(distance) / normal.length()`.

In the scalar equation of the line `ax + by = d`, this is `d`, while the `(a, b)` coordinates are represented by the normal property.

Vector2 **normal** = `Vector2(0, -1)` 🔗

- void **set_normal**(value: Vector2)
- Vector2 **get_normal**()

The line's normal, typically a unit vector. Its direction indicates the non-colliding half-plane. Can be of any length but zero. Defaults to Vector2.UP.

WorldBoundaryShape3D

Inherits: Shape3D **<** Resource **<** RefCounted **<** Object

A 3D world boundary (half-space) shape used for physics collision.

Description

A 3D world boundary shape, intended for use in physics. **WorldBoundaryShape3D** works like an infinite plane that forces all physics bodies to stay above it. The plane's normal determines which direction is considered as "above" and in the editor, the line over the plane represents this direction. It can for example be used for endless flat floors.

Properties

Plane	plane	`Plane(0, 1, 0, 0)`

Property Descriptions

Plane **plane** = `Plane(0, 1, 0, 0)` 🔗

- void **set_plane**(value: Plane)
- Plane **get_plane**()

The Plane used by the **WorldBoundaryShape3D** for collision.

X509Certificate

Inherits: Resource **<** RefCounted **<** Object

An X509 certificate (e.g. for TLS).

Description

The X509Certificate class represents an X509 certificate. Certificates can be loaded and saved like any other Resource.

They can be used as the server certificate in StreamPeerTLS.accept_stream (along with the proper CryptoKey), and to specify the only certificate that should be accepted when connecting to a TLS server via StreamPeerTLS.connect_to_stream.

Tutorials

- SSL certificates

Methods

Error	load(path: String)
Error	load_from_string(string: String)
Error	save(path: String)

String	save_to_string()

Method Descriptions

Error **load**(path: String)

Loads a certificate from `path` ("*.crt" file).

Error **load_from_string**(string: String)

Loads a certificate from the given `string`.

Error **save**(path: String)

Saves a certificate to the given `path` (should be a "*.crt" file).

String **save_to_string**()

Returns a string representation of the certificate, or an empty string if the certificate is invalid.

Object

Inherited By: AudioServer, CameraServer, ClassDB, DisplayServer, EditorFileSystemDirectory, EditorInterface, EditorPaths, EditorSelection, EditorUndoRedoManager, EditorVCSInterface, Engine, EngineDebugger, FramebufferCacheRD, GDExtensionManager, Geometry2D, Geometry3D, Input, InputMap, IP, JavaClassWrapper, JavaScriptBridge, JNISingleton, JSONRPC, MainLoop, Marshalls, MovieWriter, NativeMenu, NavigationMeshGenerator, NavigationServer2D, NavigationServer3D, Node, OpenXRExtensionWrapperExtension, OpenXRInteractionProfileMetadata, OS, Performance, PhysicsDirectBodyState2D, PhysicsDirectBodyState3D, PhysicsDirectSpaceState2D, PhysicsDirectSpaceState3D, PhysicsServer2D, PhysicsServer2DManager, PhysicsServer3D, PhysicsServer3DManager, PhysicsServer3DRenderingServerHandler, ProjectSettings, RefCounted, RenderData, RenderingDevice, RenderingServer, RenderSceneData, ResourceLoader, ResourceSaver, ResourceUID, ScriptLanguage, TextServerManager, ThemeDB, TileData, Time, TranslationServer, TreeItem, UndoRedo, UniformSetCacheRD, WorkerThreadPool, XRServer, XRVRS

Base class for all other classes in the engine.

Description

An advanced Variant type. All classes in the engine inherit from Object. Each class may define new properties, methods or signals, which are available to all inheriting classes. For example, a Sprite2D instance is able to call Node.add_child because it inherits from Node.

You can create new instances, using `Object.new()` in GDScript, or `new GodotObject` in C#.

To delete an Object instance, call free. This is necessary for most classes inheriting Object, because they do not manage memory on their own, and will otherwise cause memory leaks when no longer in use. There are a few classes that perform memory management. For example, RefCounted (and by extension Resource) deletes itself when no longer referenced, and Node deletes its children when freed.

Objects can have a Script attached to them. Once the Script is instantiated, it effectively acts as an extension to the base class, allowing it to define and inherit new properties, methods and signals.

Inside a Script, _get_property_list may be overridden to customize properties in several ways. This allows them to be available to the editor, display as lists of options, sub-divide into groups, save on disk, etc. Scripting languages offer easier ways to customize properties, such as with the @GDScript.@export annotation.

Godot is very dynamic. An object's script, and therefore its properties, methods and signals, can be changed at run-time. Because of this, there can be occasions where, for example, a property required by a method may not exist. To prevent run-time errors, see methods such as set, get, call, has_method, has_signal, etc. Note that these methods are **much** slower than direct references.

In GDScript, you can also check if a given property, method, or signal name exists in an object with the `in` operator:

```
var node = Node.new()
print("name" in node)         # Prints true
print("get_parent" in node)   # Prints true
print("tree_entered" in node) # Prints true
print("unknown" in node)      # Prints false
```

Notifications are int constants commonly sent and received by objects. For example, on every rendered frame, the SceneTree notifies nodes inside the tree with a Node.NOTIFICATION_PROCESS. The nodes receive it and may call Node._process to update. To make use of notifications, see notification and _notification.

Lastly, every object can also contain metadata (data about data). set_meta can be useful to store information that the object itself does not depend on. To keep your code clean, making excessive use of metadata is discouraged.

Note: Unlike references to a RefCounted, references to an object stored in a variable can become invalid without being set to `null`. To check if an object has been deleted, do *not* compare it against `null`. Instead, use @GlobalScope.is_instance_valid. It's also recommended to inherit from RefCounted for classes storing data instead of **Object**.

Note: The `script` is not exposed like most properties. To set or get an object's Script in code, use set_script and get_script, respectively.

Tutorials

- Object class introduction
- When and how to avoid using nodes for everything
- Object notifications

Methods

Variant	_get(property: StringName) virtual
Array[Dictionary]	_get_property_list() virtual

void	_init() virtual
void	_notification(what: int) virtual
bool	_property_can_revert(property: StringName) virtual
Variant	_property_get_revert(property: StringName) virtual
bool	_set(property: StringName, value: Variant) virtual
String	_to_string() virtual
void	_validate_property(property: Dictionary) virtual
void	add_user_signal(signal: String, arguments: Array = [])
Variant	call(method: StringName, ...) vararg
Variant	call_deferred(method: StringName, ...) vararg
Variant	callv(method: StringName, arg_array: Array)

bool	can_translate_messages() const
void	cancel_free()
Error	connect(signal: StringName, callable: Callable, flags: int = 0)
void	disconnect(signal: StringName, callable: Callable)
Error	emit_signal(signal: StringName, ...) vararg
void	free()
Variant	get(property: StringName) const
String	get_class() const
Array[Dictionary]	get_incoming_connections() const
Variant	get_indexed(property_path: NodePath) const
int	get_instance_id() const
Variant	get_meta(name: StringName, default: Variant = null) const

Array[StringName]	get_meta_list() const
int	get_method_argument_count(method: StringName) const
Array[Dictionary]	get_method_list() const
Array[Dictionary]	get_property_list() const
Variant	get_script() const
Array[Dictionary]	get_signal_connection_list(signal: StringName) const
Array[Dictionary]	get_signal_list() const
bool	has_meta(name: StringName) const
bool	has_method(method: StringName) const
bool	has_signal(signal: StringName) const
bool	has_user_signal(signal: StringName) const
bool	is_blocking_signals() const

bool	is_class(class: String) const
bool	is_connected(signal: StringName, callable: Callable) const
bool	is_queued_for_deletion() const
void	notification(what: int, reversed: bool = false)
void	notify_property_list_changed()
bool	property_can_revert(property: StringName) const
Variant	property_get_revert(property: StringName) const
void	remove_meta(name: StringName)
void	remove_user_signal(signal: StringName)
void	set(property: StringName, value: Variant)
void	set_block_signals(enable: bool)

void	set_deferred(property: StringName, value: Variant)
void	set_indexed(property_path: NodePath, value: Variant)
void	set_message_translation(enable: bool)
void	set_meta(name: StringName, value: Variant)
void	set_script(script: Variant)
String	to_string()
String	tr(message: StringName, context: StringName = &"") const
String	tr_n(message: StringName, plural_message: StringName, n: int, context: StringName = &"") const

Signals

property_list_changed() 🔗

Emitted when notify_property_list_changed is called.

script_changed()

Emitted when the object's script is changed.

Note: When this signal is emitted, the new script is not initialized yet. If you need to access the new script, defer connections to this signal with CONNECT_DEFERRED.

Enumerations

enum **ConnectFlags**:

ConnectFlags **CONNECT_DEFERRED** = `1`

Deferred connections trigger their Callables on idle time (at the end of the frame), rather than instantly.

ConnectFlags **CONNECT_PERSIST** = `2`

Persisting connections are stored when the object is serialized (such as when using PackedScene.pack). In the editor, connections created through the Node dock are always persisting.

ConnectFlags **CONNECT_ONE_SHOT** = `4`

One-shot connections disconnect themselves after emission.

ConnectFlags **CONNECT_REFERENCE_COUNTED** = `8`

Reference-counted connections can be assigned to the same Callable multiple times. Each disconnection decreases the internal counter. The signal fully disconnects only when the counter reaches 0.

Constants

NOTIFICATION_POSTINITIALIZE = `0` 🔗

Notification received when the object is initialized, before its script is attached. Used internally.

NOTIFICATION_PREDELETE = `1` 🔗

Notification received when the object is about to be deleted. Can act as the deconstructor of some programming languages.

NOTIFICATION_EXTENSION_RELOADED = `2` 🔗

Notification received when the object finishes hot reloading. This notification is only sent for extensions classes and derived.

Method Descriptions

Variant **_get**(property: StringName) virtual 🔗

Override this method to customize the behavior of get. Should return the given `property`'s value, or `null` if the `property` should be handled normally.

Combined with _set and _get_property_list, this method allows defining custom properties, which is particularly useful for editor plugins. Note that a property must be present in get_property_list, otherwise this method will not be called.

GDScript

```
func _get(property):
    if property == "fake_property":
        print("Getting my property!")
        return 4

func _get_property_list():
    return [
        { "name": "fake_property", "type": TYPE_INT }
    ]
```

C#

```
public override Variant _Get(StringName property)
{
    if (property == "FakeProperty")
    {
        GD.Print("Getting my property!");
        return 4;
    }
    return default;
}

public override
Godot.Collections.Array<Godot.Collections.Dictionary>
_GetPropertyList()
{
    return new
Godot.Collections.Array<Godot.Collections.Dictionary>()
    {
        new Godot.Collections.Dictionary()
        {
            { "name", "FakeProperty" },
            { "type", (int)Variant.Type.Int }
        }
    };
}
```

Array[Dictionary] **_get_property_list**() virtual

Override this method to provide a custom list of additional properties to handle by the engine.

Should return a property list, as an Array of dictionaries. The result is added to the array of get_property_list, and should be formatted in the same way. Each Dictionary must at least contain the `name` and `type` entries.

You can use _property_can_revert and _property_get_revert to customize the default values of the properties added by this method.

The example below displays a list of numbers shown as words going from `ZERO` to `FIVE`, with `number_count` controlling the size of the list:

GDScript

```
@tool
extends Node

@export var number_count = 3:
    set(nc):
        number_count = nc
        numbers.resize(number_count)
        notify_property_list_changed()

var numbers = PackedInt32Array([0, 0, 0])

func _get_property_list():
    var properties = []

    for i in range(number_count):
        properties.append({
            "name": "number_%d" % i,
            "type": TYPE_INT,
            "hint": PROPERTY_HINT_ENUM,
            "hint_string": "ZERO,ONE,TWO,THREE,FOUR,FIVE",
        })

    return properties

func _get(property):
    if property.begins_with("number_"):
        var index = property.get_slice("_", 1).to_int()
        return numbers[index]
```

```
func _set(property, value):
    if property.begins_with("number_"):
        var index = property.get_slice("_", 1).to_int()
        numbers[index] = value
        return true
    return false
```

C#

```
[Tool]
public partial class MyNode : Node
{
    private int _numberCount;

    [Export]
    public int NumberCount
    {
        get => _numberCount;
        set
        {
            _numberCount = value;
            _numbers.Resize(_numberCount);
            NotifyPropertyListChanged();
        }
    }

    private Godot.Collections.Array<int> _numbers = new();

    public override
Godot.Collections.Array<Godot.Collections.Dictionary>
_GetPropertyList()
    {
        var properties = new
Godot.Collections.Array<Godot.Collections.Dictionary>();

        for (int i = 0; i < _numberCount; i++)
        {
            properties.Add(new Godot.Collections.Dictionary()
            {
                { "name", $"number_{i}" },
                { "type", (int)Variant.Type.Int },
                { "hint", (int)PropertyHint.Enum },
                { "hint_string", "Zero,One,Two,Three,Four,Five"
},
            });
        }

        return properties;
    }

    public override Variant _Get(StringName property)
    {
        string propertyName = property.ToString();
        if (propertyName.StartsWith("number_"))
        {
            int index =
int.Parse(propertyName.Substring("number_".Length));
            return _numbers[index];
        }
        return default;
    }

    public override bool _Set(StringName property, Variant
value)
    {
        string propertyName = property.ToString();
        if (propertyName.StartsWith("number_"))
        {
            int index =
int.Parse(propertyName.Substring("number_".Length));
            _numbers[index] = value.As<int>();
            return true;
        }
        return false;
    }
}
```

Note: This method is intended for advanced purposes. For most common use cases, the scripting languages offer easier ways to handle properties. See @GDScript.@export, @GDScript.@export_enum, @GDScript.@export_group, etc. If you want to customize exported properties, use _validate_property.

Note: If the object's script is not @GDScript.@tool, this method will not be called in the editor.

---

void **_init**() virtual 🔗

Called when the object's script is instantiated, oftentimes after the object is initialized in memory (through `Object.new()` in GDScript, or `new GodotObject` in C#). It can be also defined to take in parameters. This method is similar to a constructor in most programming languages.

Note: If _init is defined with *required* parameters, the Object with script may only be created directly. If any other means (such as PackedScene.instantiate or Node.duplicate) are used, the script's initialization will fail.

---

void **_notification**(what: int) virtual 🔗

Called when the object receives a notification, which can be identified in `what` by comparing it with a constant. See also notification.

GDScript

```
func _notification(what):
    if what == NOTIFICATION_PREDELETE:
        print("Goodbye!")
```

C#

```
public override void _Notification(int what)
{
    if (what == NotificationPredelete)
    {
        GD.Print("Goodbye!");
    }
}
```

Note: The base **Object** defines a few notifications (NOTIFICATION_POSTINITIALIZE and NOTIFICATION_PREDELETE). Inheriting classes such as Node define a lot more notifications, which are also received by this method.

bool **_property_can_revert**(property: StringName) virtual 🔗

Override this method to customize the given `property`'s revert behavior. Should return `true` if the `property` has a custom default value and is revertible in the Inspector dock. Use _property_get_revert to specify the `property`'s default value.

Note: This method must return consistently, regardless of the current value of the `property`.

Variant **_property_get_revert**(property: StringName) virtual 🔗

Override this method to customize the given `property`'s revert behavior. Should return the default value for the `property`. If the default value differs from the `property`'s current value, a revert icon is displayed in the Inspector dock.

Note: _property_can_revert must also be overridden for this method to be called.

bool **_set**(property: StringName, value: Variant) virtual 🔗

Override this method to customize the behavior of set. Should set the `property` to `value` and return `true`, or `false` if the `property` should be handled normally. The *exact* way to set the `property` is up to this method's implementation.

Combined with _get and _get_property_list, this method allows defining custom properties, which is particularly useful for editor plugins. Note that a property *must* be present in get_property_list, otherwise this method will not be called.

GDScript

```
var internal_data = {}

func _set(property, value):
    if property == "fake_property":
        # Storing the value in the fake property.
        internal_data["fake_property"] = value
        return true
    return false

func _get_property_list():
    return [
        { "name": "fake_property", "type": TYPE_INT }
    ]
```

C#

```
private Godot.Collections.Dictionary _internalData = new
Godot.Collections.Dictionary();

public override bool _Set(StringName property, Variant value)
{
    if (property == "FakeProperty")
    {
        // Storing the value in the fake property.
        _internalData["FakeProperty"] = value;
        return true;
    }

    return false;
}
```

```
public override
Godot.Collections.Array<Godot.Collections.Dictionary>
_GetPropertyList()
{
    return new
Godot.Collections.Array<Godot.Collections.Dictionary>()
    {
        new Godot.Collections.Dictionary()
        {
            { "name", "FakeProperty" },
            { "type", (int)Variant.Type.Int }
        }
    };
}
```

String **_to_string**() virtual

Override this method to customize the return value of to_string, and therefore the object's representation as a String.

```
func _to_string():
    return "Welcome to Godot 4!"

func _init():
    print(self)       # Prints Welcome to Godot 4!"
    var a = str(self) # a is "Welcome to Godot 4!"
```

---

void **_validate_property**(property: Dictionary) virtual

Override this method to customize existing properties. Every property info goes through this method, except properties added with _get_property_list. The dictionary contents is the same as in _get_property_list.

GDScript

```
@tool
extends Node
```

```
@export var is_number_editable: bool:
    set(value):
        is_number_editable = value
        notify_property_list_changed()
@export var number: int

func _validate_property(property: Dictionary):
    if property.name == "number" and not is_number_editable:
        property.usage |= PROPERTY_USAGE_READ_ONLY
```

C#

```
[Tool]
public partial class MyNode : Node
{
    private bool _isNumberEditable;

    [Export]
    public bool IsNumberEditable
    {
        get => _isNumberEditable;
        set
        {
            _isNumberEditable = value;
            NotifyPropertyListChanged();
        }
    }

    [Export]
    public int Number { get; set; }

    public override void
_ValidateProperty(Godot.Collections.Dictionary property)
    {
        if (property["name"].AsStringName() ==
PropertyName.Number && !IsNumberEditable)
        {
            var usage =
property["usage"].As<PropertyUsageFlags>() |
PropertyUsageFlags.ReadOnly;
            property["usage"] = (int)usage;
        }
    }
}
```

---

void **add_user_signal**(signal: String, arguments: Array = []) 🔗

Adds a user-defined `signal`. Optional arguments for the signal can be added as an Array of dictionaries, each defining a `name` String and a `type` int (see Variant.Type). See also has_user_signal and remove_user_signal.

GDScript

```
add_user_signal("hurt", [
    { "name": "damage", "type": TYPE_INT },
    { "name": "source", "type": TYPE_OBJECT }
])
```

C#

```
AddUserSignal("Hurt", new Godot.Collections.Array()
{
    new Godot.Collections.Dictionary()
    {
        { "name", "damage" },
        { "type", (int)Variant.Type.Int }
    },
    new Godot.Collections.Dictionary()
    {
        { "name", "source" },
        { "type", (int)Variant.Type.Object }
    }
});
```

Variant **call**(method: StringName, ...) vararg 🔗

Calls the `method` on the object and returns the result. This method supports a variable number of arguments, so parameters can be passed as a comma separated list.

GDScript

```
var node = Node3D.new()
node.call("rotate", Vector3(1.0, 0.0, 0.0), 1.571)
```

C#

```
var node = new Node3D();
node.Call(Node3D.MethodName.Rotate, new Vector3(1f, 0f, 0f),
1.571f);
```

Note: In C#, `method` must be in snake_case when referring to built-in Godot methods. Prefer using the names exposed in the `MethodName` class to avoid allocating a new StringName on each call.

Variant **call_deferred**(method: StringName, ...) vararg 🔗

Calls the `method` on the object during idle time. Always returns null, **not** the method's result.

Idle time happens mainly at the end of process and physics frames. In it, deferred calls will be run until there are none left, which means you can defer calls from other deferred calls and they'll still be run in the current idle time cycle. This means you should not call a method deferred from itself (or from a method called by it), as this causes infinite recursion the same way as if you had called the method directly.

This method supports a variable number of arguments, so parameters can be passed as a comma separated list.

GDScript

```
var node = Node3D.new()
node.call_deferred("rotate", Vector3(1.0, 0.0, 0.0), 1.571)
```

C#

```
var node = new Node3D();
node.CallDeferred(Node3D.MethodName.Rotate, new Vector3(1f, 0f, 0f), 1.571f);
```

See also Callable.call_deferred.

Note: In C#, `method` must be in snake_case when referring to built-in Godot methods. Prefer using the names exposed in the `MethodName` class to avoid allocating a new StringName on each call.

Note: If you're looking to delay the function call by a frame, refer to the SceneTree.process_frame and SceneTree.physics_frame signals.

```
var node = Node3D.new()
# Make a Callable and bind the arguments to the node's rotate() call.
var callable = node.rotate.bind(Vector3(1.0, 0.0, 0.0), 1.571)
# Connect the callable to the process_frame signal, so it gets called in the next process frame.
# CONNECT_ONE_SHOT makes sure it only gets called once instead of every frame.
get_tree().process_frame.connect(callable, CONNECT_ONE_SHOT)
```

Variant **callv**(method: StringName, arg_array: Array) 🔗

Calls the `method` on the object and returns the result. Unlike call, this method expects all parameters to be contained inside `arg_array`.

GDScript

```
var node = Node3D.new()
node.callv("rotate", [Vector3(1.0, 0.0, 0.0), 1.571])
```

C#

```
var node = new Node3D();
node.Callv(Node3D.MethodName.Rotate, new
Godot.Collections.Array { new Vector3(1f, 0f, 0f), 1.571f });
```

Note: In C#, `method` must be in snake_case when referring to built-in Godot methods. Prefer using the names exposed in the `MethodName` class to avoid allocating a new StringName on each call.

---

bool **can_translate_messages**() const

Returns `true` if the object is allowed to translate messages with tr and tr_n. See also set_message_translation.

---

void **cancel_free**()

If this method is called during NOTIFICATION_PREDELETE, this object will reject being freed and will remain allocated. This is mostly an internal function used for error handling to avoid the user from freeing objects when they are not intended to.

Error **connect**(signal: StringName, callable: Callable, flags: int = 0)

Connects a `signal` by name to a `callable`. Optional `flags` can be also added to configure the connection's behavior (see ConnectFlags constants).

A signal can only be connected once to the same Callable. If the signal is already connected, this method returns @GlobalScope.ERR_INVALID_PARAMETER and pushes an error message, unless the signal is connected with CONNECT_REFERENCE_COUNTED. To prevent this, use is_connected first to check for existing connections.

If the `callable`'s object is freed, the connection will be lost.

Examples with recommended syntax:

Connecting signals is one of the most common operations in Godot and the API gives many options to do so, which are described further down. The code block below shows the recommended approach.

GDScript

```
func _ready():
    var button = Button.new()
    # `button_down` here is a Signal variant type, and we thus
call the Signal.connect() method, not Object.connect().
    # See discussion below for a more in-depth overview of the
API.
    button.button_down.connect(_on_button_down)

    # This assumes that a `Player` class exists, which defines
a `hit` signal.
    var player = Player.new()
    # We use Signal.connect() again, and we also use the
Callable.bind() method,
    # which returns a new Callable with the parameter binds.
    player.hit.connect(_on_player_hit.bind("sword", 100))

func _on_button_down():
    print("Button down!")

func _on_player_hit(weapon_type, damage):
    print("Hit with weapon %s for %d damage." % [weapon_type,
damage])
```

C#

```
public override void _Ready()
{
    var button = new Button();
    // C# supports passing signals as events, so we can use
this idiomatic construct:
    button.ButtonDown += OnButtonDown;

    // This assumes that a `Player` class exists, which defines a `Hit` signal.
    var player = new Player();
    // We can use lambdas when we need to bind additional parameters.
    player.Hit += () => OnPlayerHit("sword", 100);
}

private void OnButtonDown()
{
    GD.Print("Button down!");
}

private void OnPlayerHit(string weaponType, int damage)
{
    GD.Print($"Hit with weapon {weaponType} for {damage} damage.");
}
```

``Object.connect()`` or ``Signal.connect()``?

As seen above, the recommended method to connect signals is not connect. The code block below shows the four options for connecting signals, using either this legacy method or the recommended Signal.connect, and using either an implicit Callable or a manually defined one.

GDScript

```
func _ready():
    var button = Button.new()
    # Option 1: Object.connect() with an implicit Callable for the defined function.
    button.connect("button_down", _on_button_down)
    # Option 2: Object.connect() with a constructed Callable using a target object and method name.
    button.connect("button_down", Callable(self, "_on_button_down"))
    # Option 3: Signal.connect() with an implicit Callable for
the defined function.
    button.button_down.connect(_on_button_down)
    # Option 4: Signal.connect() with a constructed Callable
using a target object and method name.
    button.button_down.connect(Callable(self,
"_on_button_down"))

func _on_button_down():
    print("Button down!")
```

C#

```
public override void _Ready()
{
    var button = new Button();
    // Option 1: In C#, we can use signals as events and
connect with this idiomatic syntax:
    button.ButtonDown += OnButtonDown;
    // Option 2: GodotObject.Connect() with a constructed
Callable from a method group.
    button.Connect(Button.SignalName.ButtonDown,
Callable.From(OnButtonDown));
    // Option 3: GodotObject.Connect() with a constructed
Callable using a target object and method name.
    button.Connect(Button.SignalName.ButtonDown, new
Callable(this, MethodName.OnButtonDown));
}

private void OnButtonDown()
{
    GD.Print("Button down!");
}
```

While all options have the same outcome (`button`'s BaseButton.button_down signal will be connected to `_on_button_down`), **option 3** offers the best validation: it will print a compile-time error if either the `button_down` Signal or the `_on_button_down` Callable are not defined. On the other hand, **option 2** only relies on string names and will only be able to validate either names at runtime: it will print a runtime error if `"button_down"` doesn't correspond to a signal, or if `"_on_button_down"` is not a registered method in the object `self`. The main reason for using options 1, 2, or 4 would be if you actually need to use strings (e.g. to connect signals programmatically based on strings read from a configuration file). Otherwise, option 3 is the recommended (and fastest) method.

Binding and passing parameters:

The syntax to bind parameters is through Callable.bind, which returns a copy of the Callable with its parameters bound.

When calling emit_signal or Signal.emit, the signal parameters can be also passed. The examples below show the relationship between these signal parameters and bound parameters.

GDScript

```
func _ready():
    # This assumes that a `Player` class exists, which defines
a `hit` signal.
    var player = Player.new()
    # Using Callable.bind().
    player.hit.connect(_on_player_hit.bind("sword", 100))

    # Parameters added when emitting the signal are passed
first.
    player.hit.emit("Dark lord", 5)

# We pass two arguments when emitting (`hit_by`, `level`),
# and bind two more arguments when connecting (`weapon_type`,
`damage`).
func _on_player_hit(hit_by, level, weapon_type, damage):
    print("Hit by %s (level %d) with weapon %s for %d damage."
% [hit_by, level, weapon_type, damage])
```

C#

```
public override void _Ready()
{
    // This assumes that a `Player` class exists, which defines
a `Hit` signal.
    var player = new Player();
    // Using lambda expressions that create a closure that
captures the additional parameters.
    // The lambda only receives the parameters defined by the
signal's delegate.
    player.Hit += (hitBy, level) => OnPlayerHit(hitBy, level,
"sword", 100);

    // Parameters added when emitting the signal are passed
first.
    player.EmitSignal(SignalName.Hit, "Dark lord", 5);
}

// We pass two arguments when emitting (`hit_by`, `level`),
// and bind two more arguments when connecting (`weapon_type`,
`damage`).
private void OnPlayerHit(string hitBy, int level, string
weaponType, int damage)
{
    GD.Print($"Hit by {hitBy} (level {level}) with weapon
{weaponType} for {damage} damage.");
}
```

---

void **disconnect**(signal: StringName, callable: Callable) 🔗

Disconnects a `signal` by name from a given `callable`. If the connection does not exist, generates an error. Use is_connected to make sure that the connection exists.

Error **emit_signal**(signal: StringName, ...) vararg 🔗

Emits the given `signal` by name. The signal must exist, so it should be a built-in signal of this class or one of its inherited classes, or a user-defined signal (see add_user_signal). This method supports a variable number of arguments, so parameters can be passed as a comma separated list.

Returns @GlobalScope.ERR_UNAVAILABLE if `signal` does not exist or the parameters are invalid.

GDScript

```
emit_signal("hit", "sword", 100)
emit_signal("game_over")
```

C#

```
EmitSignal(SignalName.Hit, "sword", 100);
EmitSignal(SignalName.GameOver);
```

Note: In C#, `signal` must be in snake_case when referring to built-in Godot signals. Prefer using the names exposed in the `SignalName` class to avoid allocating a new StringName on each call.

---

void **free**() 🔗

Deletes the object from memory. Pre-existing references to the object become invalid, and any attempt to access them will result in a run-time error. Checking the references with @GlobalScope.is_instance_valid will return `false`.

Variant **get**(property: StringName) const 🔗

Returns the Variant value of the given `property`. If the `property` does not exist, this method returns `null`.

GDScript

```
var node = Node2D.new()
node.rotation = 1.5
```

```
var a = node.get("rotation") # a is 1.5
```

C#

```
var node = new Node2D();
node.Rotation = 1.5f;
var a = node.Get(Node2D.PropertyName.Rotation); // a is 1.5
```

Note: In C#, `property` must be in snake_case when referring to built-in Godot properties. Prefer using the names exposed in the `PropertyName` class to avoid allocating a new StringName on each call.

String **get_class**() const 🔗

Returns the object's built-in class name, as a String. See also is_class.

Note: This method ignores `class_name` declarations. If this object's script has defined a `class_name`, the base, built-in class name is returned instead.

Array[Dictionary] **get_incoming_connections**() const 🔗

Returns an Array of signal connections received by this object. Each connection is represented as a Dictionary that contains three entries:

- `signal` is a reference to the Signal;
- `callable` is a reference to the Callable;
- `flags` is a combination of ConnectFlags.

Variant **get_indexed**(property_path: NodePath) const 🔗

Gets the object's property indexed by the given `property_path`. The path should be a NodePath relative to the current object and can use the colon character (`:`) to access nested properties.

Examples: `"position:x"` or `"material:next_pass:blend_mode"`.

GDScript

```
var node = Node2D.new()
node.position = Vector2(5, -10)
var a = node.get_indexed("position")    # a is Vector2(5, -10)
var b = node.get_indexed("position:y") # b is -10
```

C#

```
var node = new Node2D();
node.Position = new Vector2(5, -10);
var a = node.GetIndexed("position");    // a is Vector2(5, -10)
var b = node.GetIndexed("position:y"); // b is -10
```

Note: In C#, `property_path` must be in snake_case when referring to built-in Godot properties. Prefer using the names exposed in the `PropertyName` class to avoid allocating a new StringName on each call.

Note: This method does not support actual paths to nodes in the SceneTree, only sub-property paths. In the context of nodes, use Node.get_node_and_resource instead.

---

int **get_instance_id**() const 🔗

Returns the object's unique instance ID. This ID can be saved in EncodedObjectAsID, and can be used to retrieve this object instance with @GlobalScope.instance_from_id.

Note: This ID is only useful during the current session. It won't correspond to a similar object if the ID is sent over a network, or loaded from a file at a later time.

Variant **get_meta**(name: StringName, default: Variant = null) const

Returns the object's metadata value for the given entry `name`. If the entry does not exist, returns `default`. If `default` is `null`, an error is also generated.

Note: A metadata's name must be a valid identifier as per StringName.is_valid_identifier method.

Note: Metadata that has a name starting with an underscore (_) is considered editor-only. Editor-only metadata is not displayed in the Inspector and should not be edited, although it can still be found by this method.

Array[StringName] **get_meta_list**() const

Returns the object's metadata entry names as a PackedStringArray.

---

int **get_method_argument_count**(method: StringName) const

Returns the number of arguments of the given `method` by name.

Note: In C#, `method` must be in snake_case when referring to built-in Godot methods. Prefer using the names exposed in the `MethodName` class to avoid allocating a new StringName on each call.

Array[Dictionary] **get_method_list**() const

Returns this object's methods and their signatures as an Array of dictionaries. Each Dictionary contains the following entries:

- `name` is the name of the method, as a String;
- `args` is an Array of dictionaries representing the arguments;
- `default_args` is the default arguments as an Array of variants;
- `flags` is a combination of MethodFlags;
- `id` is the method's internal identifier int;
- `return` is the returned value, as a Dictionary;

Note: The dictionaries of `args` and `return` are formatted identically to the results of get_property_list, although not all entries are used.

Array[Dictionary] **get_property_list**() const 🔗

Returns the object's property list as an Array of dictionaries. Each Dictionary contains the following entries:

- `name` is the property's name, as a String;
- `class_name` is an empty StringName, unless the property is @GlobalScope.TYPE_OBJECT and it inherits from a class;
- `type` is the property's type, as an int (see Variant.Type);
- `hint` is *how* the property is meant to be edited (see PropertyHint);
- `hint_string` depends on the hint (see PropertyHint);
- `usage` is a combination of PropertyUsageFlags.

Note: In GDScript, all class members are treated as properties. In C# and GDExtension, it may be necessary to explicitly mark class members as Godot properties using decorators or attributes.

Variant **get_script**() const 🔗

Returns the object's Script instance, or `null` if no script is attached.

Array[Dictionary] **get_signal_connection_list**(signal: StringName) const

Returns an Array of connections for the given `signal` name. Each connection is represented as a Dictionary that contains three entries:

- `signal` is a reference to the Signal;
- `callable` is a reference to the connected Callable;
- `flags` is a combination of ConnectFlags.

Array[Dictionary] **get_signal_list**() const

Returns the list of existing signals as an Array of dictionaries.

Note: Due of the implementation, each Dictionary is formatted very similarly to the returned values of get_method_list.

---

bool **has_meta**(name: StringName) const

Returns `true` if a metadata entry is found with the given `name`. See also get_meta, set_meta and remove_meta.

Note: A metadata's name must be a valid identifier as per StringName.is_valid_identifier method.

Note: Metadata that has a name starting with an underscore (_) is considered editor-only. Editor-only metadata is not displayed in the Inspector and should not be edited, although it can still be found by this method.

---

bool **has_method**(method: StringName) const

Returns `true` if the given `method` name exists in the object.

Note: In C#, `method` must be in snake_case when referring to built-in Godot methods. Prefer using the names exposed in the `MethodName` class to avoid allocating a new StringName on each call.

bool **has_signal**(signal: StringName) const

Returns `true` if the given `signal` name exists in the object.

Note: In C#, `signal` must be in snake_case when referring to built-in Godot methods. Prefer using the names exposed in the `SignalName` class to avoid allocating a new StringName on each call.

bool **has_user_signal**(signal: StringName) const

Returns `true` if the given user-defined `signal` name exists. Only signals added with add_user_signal are included. See also remove_user_signal.

bool **is_blocking_signals**() const

Returns `true` if the object is blocking its signals from being emitted. See set_block_signals.

bool **is_class**(class: String) const

Returns `true` if the object inherits from the given `class`. See also get_class.

GDScript

```
var sprite2d = Sprite2D.new()
sprite2d.is_class("Sprite2D") # Returns true
```

```
sprite2d.is_class("Node")      # Returns true
sprite2d.is_class("Node3D")    # Returns false
```

C#

```
var sprite2D = new Sprite2D();
sprite2D.IsClass("Sprite2D"); // Returns true
sprite2D.IsClass("Node");     // Returns true
sprite2D.IsClass("Node3D");   // Returns false
```

Note: This method ignores `class_name` declarations in the object's script.

---

bool **is_connected**(signal: StringName, callable: Callable) const 🔗

Returns `true` if a connection exists between the given `signal` name and `callable`.

Note: In C#, `signal` must be in snake_case when referring to built-in Godot methods. Prefer using the names exposed in the `SignalName` class to avoid allocating a new StringName on each call.

---

bool **is_queued_for_deletion**() const 🔗

Returns `true` if the Node.queue_free method was called for the object.

---

void **notification**(what: int, reversed: bool = false) 🔗

Sends the given `what` notification to all classes inherited by the object, triggering calls to _notification, starting from the highest ancestor (the **Object** class) and going down to the object's script.

If `reversed` is `true`, the call order is reversed.

GDScript

```
var player = Node2D.new()
player.set_script(load("res://player.gd"))

player.notification(NOTIFICATION_ENTER_TREE)
# The call order is Object -> Node -> Node2D -> player.gd.

player.notification(NOTIFICATION_ENTER_TREE, true)
# The call order is player.gd -> Node2D -> Node -> Object.
```

C#

```
var player = new Node2D();
player.SetScript(GD.Load("res://player.gd"));

player.Notification(NotificationEnterTree);
// The call order is GodotObject -> Node -> Node2D ->
player.gd.

player.Notification(NotificationEnterTree, true);
// The call order is player.gd -> Node2D -> Node ->
GodotObject.
```

---

void **notify_property_list_changed**() 🔗

Emits the property_list_changed signal. This is mainly used to refresh the editor, so that the Inspector and editor plugins are properly updated.

---

bool **property_can_revert**(property: StringName) const 🔗

Returns `true` if the given `property` has a custom default value. Use property_get_revert to get the `property`'s default value.

Note: This method is used by the Inspector dock to display a revert icon. The object must implement _property_can_revert to customize the default value. If _property_can_revert is not implemented, this method returns `false`.

Variant **property_get_revert**(property: StringName) const

Returns the custom default value of the given `property`. Use property_can_revert to check if the `property` has a custom default value.

Note: This method is used by the Inspector dock to display a revert icon. The object must implement _property_get_revert to customize the default value. If _property_get_revert is not implemented, this method returns `null`.

---

void **remove_meta**(name: StringName)

Removes the given entry `name` from the object's metadata. See also has_meta, get_meta and set_meta.

Note: A metadata's name must be a valid identifier as per StringName.is_valid_identifier method.

Note: Metadata that has a name starting with an underscore (_) is considered editor-only. Editor-only metadata is not displayed in the Inspector and should not be edited, although it can still be found by this method.

---

void **remove_user_signal**(signal: StringName)

Removes the given user signal `signal` from the object. See also add_user_signal and has_user_signal.

---

void **set**(property: StringName, value: Variant) 🔗

Assigns `value` to the given `property`. If the property does not exist or the given `value`'s type doesn't match, nothing happens.

GDScript

```
var node = Node2D.new()
node.set("global_scale", Vector2(8, 2.5))
print(node.global_scale) # Prints (8, 2.5)
```

C#

```
var node = new Node2D();
node.Set(Node2D.PropertyName.GlobalScale, new Vector2(8, 2.5));
GD.Print(node.GlobalScale); // Prints Vector2(8, 2.5)
```

Note: In C#, `property` must be in snake_case when referring to built-in Godot properties. Prefer using the names exposed in the `PropertyName` class to avoid allocating a new StringName on each call.

---

void **set_block_signals**(enable: bool) 🔗

If set to `true`, the object becomes unable to emit signals. As such, emit_signal and signal connections will not work, until it is set to `false`.

---

void **set_deferred**(property: StringName, value: Variant) 🔗

Assigns `value` to the given `property`, at the end of the current frame. This is equivalent to calling set through call_deferred.

GDScript

```
var node = Node2D.new()
add_child(node)

node.rotation = 1.5
node.set_deferred("rotation", 3.0)
print(node.rotation) # Prints 1.5

await get_tree().process_frame
print(node.rotation) # Prints 3.0
```

C#

```
var node = new Node2D();
node.Rotation = 1.5f;
node.SetDeferred(Node2D.PropertyName.Rotation, 3f);
GD.Print(node.Rotation); // Prints 1.5

await ToSignal(GetTree(), SceneTree.SignalName.ProcessFrame);
GD.Print(node.Rotation); // Prints 3.0
```

Note: In C#, `property` must be in snake_case when referring to built-in Godot properties. Prefer using the names exposed in the `PropertyName` class to avoid allocating a new StringName on each call.

---

void **set_indexed**(property_path: NodePath, value: Variant)

Assigns a new `value` to the property identified by the `property_path`. The path should be a NodePath relative to this object, and can use the colon character (`:`) to access nested properties.

GDScript

```
var node = Node2D.new()
node.set_indexed("position", Vector2(42, 0))
node.set_indexed("position:y", -10)
print(node.position) # Prints (42, -10)
```

C#

```
var node = new Node2D();
node.SetIndexed("position", new Vector2(42, 0));
node.SetIndexed("position:y", -10);
GD.Print(node.Position); // Prints (42, -10)
```

Note: In C#, `property_path` must be in snake_case when referring to built-in Godot properties. Prefer using the names exposed in the `PropertyName` class to avoid allocating a new StringName on each call.

---

void **set_message_translation**(enable: bool)

If set to `true`, allows the object to translate messages with tr and tr_n. Enabled by default. See also can_translate_messages.

---

void **set_meta**(name: StringName, value: Variant)

Adds or changes the entry `name` inside the object's metadata. The metadata `value` can be any Variant, although some types cannot be serialized correctly.

If `value` is `null`, the entry is removed. This is the equivalent of using remove_meta. See also has_meta and get_meta.

Note: A metadata's name must be a valid identifier as per StringName.is_valid_identifier method.

Note: Metadata that has a name starting with an underscore (_) is considered editor-only. Editor-only metadata is not displayed in the Inspector and should not be edited, although it can still be found by this method.

---

void **set_script**(script: Variant)

Attaches `script` to the object, and instantiates it. As a result, the script's _init is called. A Script is used to extend the object's functionality.

If a script already exists, its instance is detached, and its property values and state are lost. Built-in property values are still kept.

String **to_string**() 🔗

Returns a String representing the object. Defaults to `"<ClassName#RID>"`. Override _to_string to customize the string representation of the object.

String **tr**(message: StringName, context: StringName = &"") const 🔗

Translates a `message`, using the translation catalogs configured in the Project Settings. Further `context` can be specified to help with the translation. Note that most Control nodes automatically translate their strings, so this method is mostly useful for formatted strings or custom drawn text.

If can_translate_messages is `false`, or no translation is available, this method returns the `message` without changes. See set_message_translation.

For detailed examples, see Internationalizing games.

Note: This method can't be used without an **Object** instance, as it requires the can_translate_messages method. To translate strings in a static context, use TranslationServer.translate.

String **tr_n**(message: StringName, plural_message: StringName, n: int, context: StringName = &"") const 🔗

Translates a `message` or `plural_message`, using the translation catalogs configured in the Project Settings. Further `context` can be specified to help with the translation.

If can_translate_messages is `false`, or no translation is available, this method returns `message` or `plural_message`, without changes. See set_message_translation.

The `n` is the number, or amount, of the message's subject. It is used by the translation system to fetch the correct plural form for the current language.

For detailed examples, see Localization using gettext.

Note: Negative and float numbers may not properly apply to some countable subjects. It's recommended to handle these cases with tr.

Note: This method can't be used without an **Object** instance, as it requires the can_translate_messages method. To translate strings in a static context, use TranslationServer.translate_plural.

AESContext

Inherits: RefCounted **<** Object

Provides access to AES encryption/decryption of raw data.

Description

This class holds the context information required for encryption and decryption operations with AES (Advanced Encryption Standard). Both AES-ECB and AES-CBC modes are supported.

GDScript

```
extends Node

var aes = AESContext.new()

func _ready():
    var key = "My secret key!!!" # Key must be either 16 or 32
bytes.
    var data = "My secret text!!" # Data size must be multiple
of 16 bytes, apply padding if needed.
    # Encrypt ECB
    aes.start(AESContext.MODE_ECB_ENCRYPT,
key.to_utf8_buffer())
    var encrypted = aes.update(data.to_utf8_buffer())
    aes.finish()
    # Decrypt ECB
    aes.start(AESContext.MODE_ECB_DECRYPT,
key.to_utf8_buffer())
    var decrypted = aes.update(encrypted)
    aes.finish()
    # Check ECB
    assert(decrypted == data.to_utf8_buffer())

    var iv = "My secret iv!!!!" # IV must be of exactly 16
```

```
bytes.
    # Encrypt CBC
    aes.start(AESContext.MODE_CBC_ENCRYPT,
key.to_utf8_buffer(), iv.to_utf8_buffer())
    encrypted = aes.update(data.to_utf8_buffer())
    aes.finish()
    # Decrypt CBC
    aes.start(AESContext.MODE_CBC_DECRYPT,
key.to_utf8_buffer(), iv.to_utf8_buffer())
    decrypted = aes.update(encrypted)
    aes.finish()
    # Check CBC
    assert(decrypted == data.to_utf8_buffer())
```

C#

```
using Godot;
using System.Diagnostics;

public partial class MyNode : Node
{
    private AesContext _aes = new AesContext();

    public override void _Ready()
    {
        string key = "My secret key!!!"; // Key must be either
16 or 32 bytes.
        string data = "My secret text!!"; // Data size must be
multiple of 16 bytes, apply padding if needed.
        // Encrypt ECB
        _aes.Start(AesContext.Mode.EcbEncrypt,
key.ToUtf8Buffer());
        byte[] encrypted = _aes.Update(data.ToUtf8Buffer());
        _aes.Finish();
        // Decrypt ECB
        _aes.Start(AesContext.Mode.EcbDecrypt,
key.ToUtf8Buffer());
        byte[] decrypted = _aes.Update(encrypted);
        _aes.Finish();
        // Check ECB
        Debug.Assert(decrypted == data.ToUtf8Buffer());
        string iv = "My secret iv!!!!"; // IV must be of
exactly 16 bytes.
        // Encrypt CBC
        _aes.Start(AesContext.Mode.EcbEncrypt,
key.ToUtf8Buffer(), iv.ToUtf8Buffer());
        encrypted = _aes.Update(data.ToUtf8Buffer());
        _aes.Finish();
        // Decrypt CBC
        _aes.Start(AesContext.Mode.EcbDecrypt,
key.ToUtf8Buffer(), iv.ToUtf8Buffer());
        decrypted = _aes.Update(encrypted);
        _aes.Finish();
        // Check CBC
        Debug.Assert(decrypted == data.ToUtf8Buffer());
    }
}
```

Methods

void	finish()
PackedByteArray	get_iv_state()
Error	start(mode: Mode, key: PackedByteArray, iv: PackedByteArray = PackedByteArray())
PackedByteArray	update(src: PackedByteArray)

Enumerations

enum **Mode:** 🔗

Mode **MODE_ECB_ENCRYPT** = `0`

AES electronic codebook encryption mode.

Mode **MODE_ECB_DECRYPT** = `1`

AES electronic codebook decryption mode.

Mode **MODE_CBC_ENCRYPT** = `2`

AES cipher blocker chaining encryption mode.

Mode **MODE_CBC_DECRYPT** = `3`

AES cipher blocker chaining decryption mode.

Mode **MODE_MAX** = `4`

Maximum value for the mode enum.

Method Descriptions

void **finish**() 🔗

Close this AES context so it can be started again. See start.

PackedByteArray **get_iv_state**() 🔗

Get the current IV state for this context (IV gets updated when calling update). You normally don't need this function.

Note: This function only makes sense when the context is started with MODE_CBC_ENCRYPT or MODE_CBC_DECRYPT.

Error **start**(mode: Mode, key: PackedByteArray, iv: PackedByteArray = PackedByteArray())

Start the AES context in the given `mode`. A `key` of either 16 or 32 bytes must always be provided, while an `iv` (initialization vector) of exactly 16 bytes, is only needed when `mode` is either MODE_CBC_ENCRYPT or MODE_CBC_DECRYPT.

PackedByteArray **update**(src: PackedByteArray)

Run the desired operation for this AES context. Will return a PackedByteArray containing the result of encrypting (or decrypting) the given `src`. See start for mode of operation.

Note: The size of `src` must be a multiple of 16. Apply some padding if needed.

AStar2D

Inherits: RefCounted **<** Object

An implementation of A* for finding the shortest path between two vertices on a connected graph in 2D space.

Description

An implementation of the A* algorithm, used to find the shortest path between two vertices on a connected graph in 2D space.

See AStar3D for a more thorough explanation on how to use this class. **AStar2D** is a wrapper for AStar3D that enforces 2D coordinates.

Methods

float	_compute_cost(from_id: int, to_id: int) virtual const
float	_estimate_cost(from_id: int, to_id: int) virtual const
void	add_point(id: int, position: Vector2, weight_scale: float = 1.0)

bool	are_points_connected(id: int, to_id: int, bidirectional: bool = true) const
void	clear()
void	connect_points(id: int, to_id: int, bidirectional: bool = true)
void	disconnect_points(id: int, to_id: int, bidirectional: bool = true)
int	get_available_point_id() const
int	get_closest_point(to_position: Vector2, include_disabled: bool = false) const
Vector2	get_closest_position_in_segment(to_position: Vector2) const
PackedInt64Array	get_id_path(from_id: int, to_id: int, allow_partial_path: bool = false)
int	get_point_capacity() const
PackedInt64Array	get_point_connections(id: int)
int	get_point_count() const

PackedInt64Array	get_point_ids()
PackedVector2Array	get_point_path(from_id: int, to_id: int, allow_partial_path: bool = false)
Vector2	get_point_position(id: int) const
float	get_point_weight_scale(id: int) const
bool	has_point(id: int) const
bool	is_point_disabled(id: int) const
void	remove_point(id: int)
void	reserve_space(num_nodes: int)
void	set_point_disabled(id: int, disabled: bool = true)
void	set_point_position(id: int, position: Vector2)
void	set_point_weight_scale(id: int, weight_scale: float)

Method Descriptions

float **_compute_cost**(from_id: int, to_id: int) virtual const 🔗

Called when computing the cost between two connected points.

Note that this function is hidden in the default **AStar2D** class.

---

float **_estimate_cost**(from_id: int, to_id: int) virtual const 🔗

Called when estimating the cost between a point and the path's ending point.

Note that this function is hidden in the default **AStar2D** class.

---

void **add_point**(id: int, position: Vector2, weight_scale: float = 1.0) 🔗

Adds a new point at the given position with the given identifier. The `id` must be 0 or larger, and the `weight_scale` must be 0.0 or greater.

The `weight_scale` is multiplied by the result of _compute_cost when determining the overall cost of traveling across a segment from a neighboring point to this point. Thus, all else being equal, the algorithm prefers points with lower `weight_scale`s to form a path.

GDScript

```
var astar = AStar2D.new()
astar.add_point(1, Vector2(1, 0), 4) # Adds the point (1, 0) with weight_scale 4 and id 1
```

C#

```
var astar = new AStar2D();
astar.AddPoint(1, new Vector2(1, 0), 4); // Adds the point (1,
0) with weight_scale 4 and id 1
```

If there already exists a point for the given `id`, its position and weight scale are updated to the given values.

---

bool **are_points_connected**(id: int, to_id: int, bidirectional: bool = true) const

Returns whether there is a connection/segment between the given points. If `bidirectional` is `false`, returns whether movement from `id` to `to_id` is possible through this segment.

---

void **clear**()

Clears all the points and segments.

---

void **connect_points**(id: int, to_id: int, bidirectional: bool = true)

Creates a segment between the given points. If `bidirectional` is `false`, only movement from `id` to `to_id` is allowed, not the reverse direction.

GDScript

```
var astar = AStar2D.new()
astar.add_point(1, Vector2(1, 1))
astar.add_point(2, Vector2(0, 5))
astar.connect_points(1, 2, false)
```

C#

```
var astar = new AStar2D();
astar.AddPoint(1, new Vector2(1, 1));
astar.AddPoint(2, new Vector2(0, 5));
astar.ConnectPoints(1, 2, false);
```

---

void **disconnect_points**(id: int, to_id: int, bidirectional: bool = true) 🔗

Deletes the segment between the given points. If `bidirectional` is `false`, only movement from `id` to `to_id` is prevented, and a unidirectional segment possibly remains.

---

int **get_available_point_id**() const 🔗

Returns the next available point ID with no point associated to it.

---

int **get_closest_point**(to_position: Vector2, include_disabled: bool = false) const 🔗

Returns the ID of the closest point to `to_position`, optionally taking disabled points into account. Returns `-1` if there are no points in the points pool.

Note: If several points are the closest to `to_position`, the one with the smallest ID will be returned, ensuring a deterministic result.

Vector2 **get_closest_position_in_segment**(to_position: Vector2) const 🔗

Returns the closest position to `to_position` that resides inside a segment between two connected points.

GDScript

```
var astar = AStar2D.new()
astar.add_point(1, Vector2(0, 0))
astar.add_point(2, Vector2(0, 5))
astar.connect_points(1, 2)
var res = astar.get_closest_position_in_segment(Vector2(3, 3))
# Returns (0, 3)
```

C#

```
var astar = new AStar2D();
astar.AddPoint(1, new Vector2(0, 0));
astar.AddPoint(2, new Vector2(0, 5));
astar.ConnectPoints(1, 2);
Vector2 res = astar.GetClosestPositionInSegment(new Vector2(3,
3)); // Returns (0, 3)
```

The result is in the segment that goes from `y = 0` to `y = 5`. It's the closest position in the segment to the given point.

PackedInt64Array **get_id_path**(from_id: int, to_id: int, allow_partial_path: bool = false) 🔗

Returns an array with the IDs of the points that form the path found by AStar2D between the given points. The array is ordered from the starting point to the ending point of the path.

If there is no valid path to the target, and `allow_partial_path` is `true`, returns a path to the point closest to the target that can be reached.

GDScript

```
var astar = AStar2D.new()
astar.add_point(1, Vector2(0, 0))
astar.add_point(2, Vector2(0, 1), 1) # Default weight is 1
astar.add_point(3, Vector2(1, 1))
astar.add_point(4, Vector2(2, 0))

astar.connect_points(1, 2, false)
astar.connect_points(2, 3, false)
astar.connect_points(4, 3, false)
astar.connect_points(1, 4, false)

var res = astar.get_id_path(1, 3) # Returns [1, 2, 3]
```

C#

```
var astar = new AStar2D();
astar.AddPoint(1, new Vector2(0, 0));
astar.AddPoint(2, new Vector2(0, 1), 1); // Default weight is 1
astar.AddPoint(3, new Vector2(1, 1));
astar.AddPoint(4, new Vector2(2, 0));

astar.ConnectPoints(1, 2, false);
astar.ConnectPoints(2, 3, false);
astar.ConnectPoints(4, 3, false);
astar.ConnectPoints(1, 4, false);
long[] res = astar.GetIdPath(1, 3); // Returns [1, 2, 3]
```

If you change the 2nd point's weight to 3, then the result will be `[1, 4, 3]` instead, because now even though the distance is longer, it's "easier" to get through point 4 than through point 2.

---

int **get_point_capacity**() const 🔗

Returns the capacity of the structure backing the points, useful in conjunction with reserve_space.

PackedInt64Array **get_point_connections**(id: int) 🔗

Returns an array with the IDs of the points that form the connection with the given point.

GDScript

```
var astar = AStar2D.new()
astar.add_point(1, Vector2(0, 0))
astar.add_point(2, Vector2(0, 1))
astar.add_point(3, Vector2(1, 1))
astar.add_point(4, Vector2(2, 0))

astar.connect_points(1, 2, true)
astar.connect_points(1, 3, true)

var neighbors = astar.get_point_connections(1) # Returns [2, 3]
```

C#

```
var astar = new AStar2D();
astar.AddPoint(1, new Vector2(0, 0));
astar.AddPoint(2, new Vector2(0, 1));
astar.AddPoint(3, new Vector2(1, 1));
astar.AddPoint(4, new Vector2(2, 0));

astar.ConnectPoints(1, 2, true);
astar.ConnectPoints(1, 3, true);

long[] neighbors = astar.GetPointConnections(1); // Returns [2,
3]
```

---

int **get_point_count**() const

Returns the number of points currently in the points pool.

PackedInt64Array **get_point_ids**()

Returns an array of all point IDs.

PackedVector2Array **get_point_path**(from_id: int, to_id: int, allow_partial_path: bool = false)

Returns an array with the points that are in the path found by AStar2D between the given points. The array is ordered from the starting point to the ending point of the path.

If there is no valid path to the target, and `allow_partial_path` is `true`, returns a path to the point closest to the target that can be reached.

Note: This method is not thread-safe. If called from a Thread, it will return an empty array and will print an error message.

Vector2 **get_point_position**(id: int) const

Returns the position of the point associated with the given `id`.

---

float **get_point_weight_scale**(id: int) const

Returns the weight scale of the point associated with the given `id`.

---

bool **has_point**(id: int) const

Returns whether a point associated with the given `id` exists.

---

bool **is_point_disabled**(id: int) const

Returns whether a point is disabled or not for pathfinding. By default, all points are enabled.

---

void **remove_point**(id: int)

Removes the point associated with the given `id` from the points pool.

---

void **reserve_space**(num_nodes: int) 🔗

Reserves space internally for `num_nodes` points, useful if you're adding a known large number of points at once, such as points on a grid. New capacity must be greater or equals to old capacity.

---

void **set_point_disabled**(id: int, disabled: bool = true) 🔗

Disables or enables the specified point for pathfinding. Useful for making a temporary obstacle.

---

void **set_point_position**(id: int, position: Vector2) 🔗

Sets the `position` for the point with the given `id`.

---

void **set_point_weight_scale**(id: int, weight_scale: float) 🔗

Sets the `weight_scale` for the point with the given `id`. The `weight_scale` is multiplied by the result of _compute_cost when determining the overall cost of traveling across a segment from a neighboring point to this point.

AStar3D

Inherits: RefCounted **<** Object

An implementation of A* for finding the shortest path between two vertices on a connected graph in 3D space.

Description

A* (A star) is a computer algorithm used in pathfinding and graph traversal, the process of plotting short paths among vertices (points), passing through a given set of edges (segments). It enjoys widespread use due to its performance and accuracy. Godot's A* implementation uses points in 3D space and Euclidean distances by default.

You must add points manually with add_point and create segments manually with connect_points. Once done, you can test if there is a path between two points with the are_points_connected function, get a path containing indices by get_id_path, or one containing actual coordinates with get_point_path.

It is also possible to use non-Euclidean distances. To do so, create a class that extends **AStar3D** and override methods _compute_cost and _estimate_cost. Both take two indices and return a length, as is shown in the following example.

GDScript

```
class MyAStar:
    extends AStar3D

    func _compute_cost(u, v):
        return abs(u - v)
    func _estimate_cost(u, v):
        return min(0, abs(u - v) - 1)
```

C#

```
public partial class MyAStar : AStar3D
{
    public override float _ComputeCost(long fromId, long toId)
    {
        return Mathf.Abs((int)(fromId - toId));
    }

    public override float _EstimateCost(long fromId, long toId)
    {
        return Mathf.Min(0, Mathf.Abs((int)(fromId - toId)) -
1);
    }
}
```

_estimate_cost should return a lower bound of the distance, i.e. `_estimate_cost(u, v) <= _compute_cost(u, v)`. This serves as a hint to the algorithm because the custom _compute_cost might be computation-heavy. If this is not the case, make _estimate_cost return the same value as _compute_cost to provide the algorithm with the most accurate information.

If the default _estimate_cost and _compute_cost methods are used, or if the supplied _estimate_cost method returns a lower bound of the cost, then the paths returned by A* will be the lowest-cost paths. Here, the cost of a path equals the sum of the _compute_cost results of all segments in the path multiplied by the `weight_scale`s of the endpoints of the respective segments. If the default methods are used and the `weight_scale`s of all points are set to `1.0`, then this equals the sum of Euclidean distances of all segments in the path.

Methods

float	_compute_cost(from_id: int, to_id: int) virtual const
float	_estimate_cost(from_id: int, to_id: int) virtual const
void	add_point(id: int, position: Vector3, weight_scale: float = 1.0)
bool	are_points_connected(id: int, to_id: int, bidirectional: bool = true) const
void	clear()
void	connect_points(id: int, to_id: int, bidirectional: bool = true)
void	disconnect_points(id: int, to_id: int, bidirectional: bool = true)
int	get_available_point_id() const
int	get_closest_point(to_position: Vector3, include_disabled: bool = false) const
Vector3	get_closest_position_in_segment(to_position: Vector3) const

PackedInt64Array	get_id_path(from_id: int, to_id: int, allow_partial_path: bool = false)
int	get_point_capacity() const
PackedInt64Array	get_point_connections(id: int)
int	get_point_count() const
PackedInt64Array	get_point_ids()
PackedVector3Array	get_point_path(from_id: int, to_id: int, allow_partial_path: bool = false)
Vector3	get_point_position(id: int) const
float	get_point_weight_scale(id: int) const
bool	has_point(id: int) const
bool	is_point_disabled(id: int) const
void	remove_point(id: int)
void	reserve_space(num_nodes: int)

void	set_point_disabled(id: int, disabled: bool = true)
void	set_point_position(id: int, position: Vector3)
void	set_point_weight_scale(id: int, weight_scale: float)

Method Descriptions

float **_compute_cost**(from_id: int, to_id: int) virtual const 🔗

Called when computing the cost between two connected points.

Note that this function is hidden in the default **AStar3D** class.

float **_estimate_cost**(from_id: int, to_id: int) virtual const 🔗

Called when estimating the cost between a point and the path's ending point.

Note that this function is hidden in the default **AStar3D** class.

void **add_point**(id: int, position: Vector3, weight_scale: float = 1.0) 🔗

Adds a new point at the given position with the given identifier. The `id` must be 0 or larger, and the `weight_scale` must be 0.0 or greater.

The `weight_scale` is multiplied by the result of _compute_cost when determining the overall cost of traveling across a segment from a neighboring point to this point. Thus, all else being equal, the algorithm prefers points with lower `weight_scale`s to form a path.

GDScript

```
var astar = AStar3D.new()
astar.add_point(1, Vector3(1, 0, 0), 4) # Adds the point (1, 0, 0) with weight_scale 4 and id 1
```

C#

```
var astar = new AStar3D();
astar.AddPoint(1, new Vector3(1, 0, 0), 4); // Adds the point (1, 0, 0) with weight_scale 4 and id 1
```

If there already exists a point for the given `id`, its position and weight scale are updated to the given values.

---

bool **are_points_connected**(id: int, to_id: int, bidirectional: bool = true) const

Returns whether the two given points are directly connected by a segment. If `bidirectional` is `false`, returns whether movement from `id` to `to_id` is possible through this segment.

---

void **clear**()

Clears all the points and segments.

---

void **connect_points**(id: int, to_id: int, bidirectional: bool = true)

Creates a segment between the given points. If `bidirectional` is `false`, only movement from `id` to `to_id` is allowed, not the reverse direction.

GDScript

```
var astar = AStar3D.new()
astar.add_point(1, Vector3(1, 1, 0))
astar.add_point(2, Vector3(0, 5, 0))
astar.connect_points(1, 2, false)
```

C#

```
var astar = new AStar3D();
astar.AddPoint(1, new Vector3(1, 1, 0));
astar.AddPoint(2, new Vector3(0, 5, 0));
astar.ConnectPoints(1, 2, false);
```

---

void **disconnect_points**(id: int, to_id: int, bidirectional: bool = true)

Deletes the segment between the given points. If `bidirectional` is `false`, only movement from `id` to `to_id` is prevented, and a unidirectional segment possibly remains.

---

int **get_available_point_id**() const

Returns the next available point ID with no point associated to it.

---

int **get_closest_point**(to_position: Vector3, include_disabled: bool = false) const

Returns the ID of the closest point to `to_position`, optionally taking disabled points into account. Returns `-1` if there are no points in the points pool.

Note: If several points are the closest to `to_position`, the one with the smallest ID will be returned, ensuring a deterministic result.

Vector3 **get_closest_position_in_segment**(to_position: Vector3) const 🔗

Returns the closest position to `to_position` that resides inside a segment between two connected points.

GDScript

```
var astar = AStar3D.new()
astar.add_point(1, Vector3(0, 0, 0))
astar.add_point(2, Vector3(0, 5, 0))
astar.connect_points(1, 2)
var res = astar.get_closest_position_in_segment(Vector3(3, 3,
0)) # Returns (0, 3, 0)
```

C#

```
var astar = new AStar3D();
astar.AddPoint(1, new Vector3(0, 0, 0));
astar.AddPoint(2, new Vector3(0, 5, 0));
astar.ConnectPoints(1, 2);
Vector3 res = astar.GetClosestPositionInSegment(new Vector3(3,
3, 0)); // Returns (0, 3, 0)
```

The result is in the segment that goes from `y = 0` to `y = 5`. It's the closest position in the segment to the given point.

PackedInt64Array **get_id_path**(from_id: int, to_id: int, allow_partial_path: bool = false) 🔗

Returns an array with the IDs of the points that form the path found by AStar3D between the given points. The array is ordered from the starting point to the ending point of the path.

If there is no valid path to the target, and `allow_partial_path` is `true`, returns a path to the point closest to the target that can be reached.

GDScript

```
var astar = AStar3D.new()
astar.add_point(1, Vector3(0, 0, 0))
astar.add_point(2, Vector3(0, 1, 0), 1) # Default weight is 1
astar.add_point(3, Vector3(1, 1, 0))
astar.add_point(4, Vector3(2, 0, 0))

astar.connect_points(1, 2, false)
astar.connect_points(2, 3, false)
astar.connect_points(4, 3, false)
astar.connect_points(1, 4, false)

var res = astar.get_id_path(1, 3) # Returns [1, 2, 3]
```

C#

```
var astar = new AStar3D();
astar.AddPoint(1, new Vector3(0, 0, 0));
astar.AddPoint(2, new Vector3(0, 1, 0), 1); // Default weight
is 1
astar.AddPoint(3, new Vector3(1, 1, 0));
astar.AddPoint(4, new Vector3(2, 0, 0));
astar.ConnectPoints(1, 2, false);
astar.ConnectPoints(2, 3, false);
astar.ConnectPoints(4, 3, false);
astar.ConnectPoints(1, 4, false);
long[] res = astar.GetIdPath(1, 3); // Returns [1, 2, 3]
```

If you change the 2nd point's weight to 3, then the result will be `[1, 4, 3]` instead, because now even though the distance is longer, it's "easier" to get through point 4 than through point 2.

---

int **get_point_capacity**() const 🔗

Returns the capacity of the structure backing the points, useful in conjunction with reserve_space.

PackedInt64Array **get_point_connections**(id: int) 🔗

Returns an array with the IDs of the points that form the connection with the given point.

GDScript

```
var astar = AStar3D.new()
astar.add_point(1, Vector3(0, 0, 0))
astar.add_point(2, Vector3(0, 1, 0))
astar.add_point(3, Vector3(1, 1, 0))
astar.add_point(4, Vector3(2, 0, 0))

astar.connect_points(1, 2, true)
astar.connect_points(1, 3, true)

var neighbors = astar.get_point_connections(1) # Returns [2, 3]
```

C#

```
var astar = new AStar3D();
astar.AddPoint(1, new Vector3(0, 0, 0));
astar.AddPoint(2, new Vector3(0, 1, 0));
astar.AddPoint(3, new Vector3(1, 1, 0));
astar.AddPoint(4, new Vector3(2, 0, 0));
astar.ConnectPoints(1, 2, true);
astar.ConnectPoints(1, 3, true);

long[] neighbors = astar.GetPointConnections(1); // Returns [2, 3]
```

---

int **get_point_count**() const 🔗

Returns the number of points currently in the points pool.

PackedInt64Array **get_point_ids**()

Returns an array of all point IDs.

PackedVector3Array **get_point_path**(from_id: int, to_id: int, allow_partial_path: bool = false)

Returns an array with the points that are in the path found by AStar3D between the given points. The array is ordered from the starting point to the ending point of the path.

If there is no valid path to the target, and `allow_partial_path` is `true`, returns a path to the point closest to the target that can be reached.

Note: This method is not thread-safe. If called from a Thread, it will return an empty array and will print an error message.

Vector3 **get_point_position**(id: int) const

Returns the position of the point associated with the given `id`.

float **get_point_weight_scale**(id: int) const

Returns the weight scale of the point associated with the given `id`.

bool **has_point**(id: int) const

Returns whether a point associated with the given `id` exists.

bool **is_point_disabled**(id: int) const

Returns whether a point is disabled or not for pathfinding. By default, all points are enabled.

---

void **remove_point**(id: int) 🔗

Removes the point associated with the given `id` from the points pool.

---

void **reserve_space**(num_nodes: int) 🔗

Reserves space internally for `num_nodes` points. Useful if you're adding a known large number of points at once, such as points on a grid. New capacity must be greater or equals to old capacity.

---

void **set_point_disabled**(id: int, disabled: bool = true) 🔗

Disables or enables the specified point for pathfinding. Useful for making a temporary obstacle.

---

void **set_point_position**(id: int, position: Vector3) 🔗

Sets the `position` for the point with the given `id`.

---

void **set_point_weight_scale**(id: int, weight_scale: float) 🔗

Sets the `weight_scale` for the point with the given `id`. The `weight_scale` is multiplied by the result of _compute_cost when determining the overall cost of traveling across a segment from a neighboring point to this point.

AStarGrid2D

Inherits: RefCounted < Object

An implementation of A* for finding the shortest path between two points on a partial 2D grid.

Description

AStarGrid2D is a variant of AStar2D that is specialized for partial 2D grids. It is simpler to use because it doesn't require you to manually create points and connect them together. This class also supports multiple types of heuristics, modes for diagonal movement, and a jumping mode to speed up calculations.

To use **AStarGrid2D**, you only need to set the region of the grid, optionally set the cell_size, and then call the update method:

GDScript

```
var astar_grid = AStarGrid2D.new()
astar_grid.region = Rect2i(0, 0, 32, 32)
astar_grid.cell_size = Vector2(16, 16)
astar_grid.update()
print(astar_grid.get_id_path(Vector2i(0, 0), Vector2i(3, 4))) # prints (0, 0), (1, 1), (2, 2), (3, 3), (3, 4)
print(astar_grid.get_point_path(Vector2i(0, 0), Vector2i(3, 4))) # prints (0, 0), (16, 16), (32, 32), (48, 48), (48, 64)
```

C#

```
AStarGrid2D astarGrid = new AStarGrid2D();
astarGrid.Region = new Rect2I(0, 0, 32, 32);
astarGrid.CellSize = new Vector2I(16, 16);
astarGrid.Update();
```

```
GD.Print(astarGrid.GetIdPath(Vector2I.Zero, new Vector2I(3,
4))); // prints (0, 0), (1, 1), (2, 2), (3, 3), (3, 4)
GD.Print(astarGrid.GetPointPath(Vector2I.Zero, new Vector2I(3,
4))); // prints (0, 0), (16, 16), (32, 32), (48, 48), (48, 64)
```

To remove a point from the pathfinding grid, it must be set as "solid" with set_point_solid.

Properties

CellShape	cell_shape	`0`
Vector2	cell_size	`Vector2(1, 1)`
Heuristic	default_compute_heuristic	`0`
Heuristic	default_estimate_heuristic	`0`
DiagonalMode	diagonal_mode	`0`
bool	jumping_enabled	`false`
Vector2	offset	`Vector2(0, 0)`
Rect2i	region	`Rect2i(0, 0, 0, 0)`
Vector2i	size	`Vector2i(0, 0)`

Methods

float	_compute_cost(from_id: Vector2i, to_id: Vector2i) virtual const
float	_estimate_cost(from_id: Vector2i, to_id: Vector2i) virtual const
void	clear()
void	fill_solid_region(region: Rect2i, solid: bool = true)
void	fill_weight_scale_region(region: Rect2i, weight_scale: float)
Array[Vector2i]	get_id_path(from_id: Vector2i, to_id: Vector2i, allow_partial_path: bool = false)
PackedVector2Array	get_point_path(from_id: Vector2i, to_id: Vector2i, allow_partial_path: bool = false)
Vector2	get_point_position(id: Vector2i) const
float	get_point_weight_scale(id: Vector2i) const

bool	is_dirty() const
bool	is_in_bounds(x: int, y: int) const
bool	is_in_boundsv(id: Vector2i) const
bool	is_point_solid(id: Vector2i) const
void	set_point_solid(id: Vector2i, solid: bool = true)
void	set_point_weight_scale(id: Vector2i, weight_scale: float)
void	update()

Enumerations

enum **Heuristic**: 🔗

Heuristic **HEURISTIC_EUCLIDEAN** = `0`

The Euclidean heuristic [https://en.wikipedia.org/wiki/Euclidean_distance] to be used for the pathfinding using the following formula:

```
dx = abs(to_id.x - from_id.x)
dy = abs(to_id.y - from_id.y)
result = sqrt(dx * dx + dy * dy)
```

Note: This is also the internal heuristic used in AStar3D and AStar2D by default (with the inclusion of possible z-axis coordinate).

Heuristic **HEURISTIC_MANHATTAN** = `1`

The Manhattan heuristic [https://en.wikipedia.org/wiki/Taxicab_geometry] to be used for the pathfinding using the following formula:

```
dx = abs(to_id.x - from_id.x)
dy = abs(to_id.y - from_id.y)
result = dx + dy
```

Note: This heuristic is intended to be used with 4-side orthogonal movements, provided by setting the diagonal_mode to DIAGONAL_MODE_NEVER.

Heuristic **HEURISTIC_OCTILE** = `2`

The Octile heuristic to be used for the pathfinding using the following formula:

```
dx = abs(to_id.x - from_id.x)
dy = abs(to_id.y - from_id.y)
f = sqrt(2) - 1
result = (dx < dy) ? f * dx + dy : f * dy + dx;
```

Heuristic **HEURISTIC_CHEBYSHEV** = `3`

The Chebyshev heuristic [https://en.wikipedia.org/wiki/Chebyshev_distance] to be used for the pathfinding using the following formula:

```
dx = abs(to_id.x - from_id.x)
dy = abs(to_id.y - from_id.y)
result = max(dx, dy)
```

Heuristic **HEURISTIC_MAX** = `4`

Represents the size of the Heuristic enum.

---

enum **DiagonalMode**: 🔗

DiagonalMode **DIAGONAL_MODE_ALWAYS** = `0`

The pathfinding algorithm will ignore solid neighbors around the target cell and allow passing using diagonals.

DiagonalMode **DIAGONAL_MODE_NEVER** = `1`

The pathfinding algorithm will ignore all diagonals and the way will be always orthogonal.

DiagonalMode **DIAGONAL_MODE_AT_LEAST_ONE_WALKABLE** = `2`

The pathfinding algorithm will avoid using diagonals if at least two obstacles have been placed around the neighboring cells of the specific path segment.

DiagonalMode **DIAGONAL_MODE_ONLY_IF_NO_OBSTACLES** = `3`

The pathfinding algorithm will avoid using diagonals if any obstacle has been placed around the neighboring cells of the specific path segment.

DiagonalMode **DIAGONAL_MODE_MAX** = `4`

Represents the size of the DiagonalMode enum.

---

enum **CellShape**: 🔗

CellShape **CELL_SHAPE_SQUARE** = `0`

Rectangular cell shape.

CellShape **CELL_SHAPE_ISOMETRIC_RIGHT** = `1`

Diamond cell shape (for isometric look). Cell coordinates layout where the horizontal axis goes up-right, and the vertical one goes down-right.

CellShape **CELL_SHAPE_ISOMETRIC_DOWN** = `2`

Diamond cell shape (for isometric look). Cell coordinates layout where the horizontal axis goes down-right, and the vertical one goes down-left.

CellShape **CELL_SHAPE_MAX** = `3`

Represents the size of the CellShape enum.

Property Descriptions

CellShape **cell_shape** = `0` 🔗

- void **set_cell_shape**(value: CellShape)
- CellShape **get_cell_shape**()

The cell shape. Affects how the positions are placed in the grid. If changed, update needs to be called before finding the next path.

Vector2 **cell_size** = `Vector2(1, 1)` 🔗

- void **set_cell_size**(value: Vector2)
- Vector2 **get_cell_size**()

The size of the point cell which will be applied to calculate the resulting point position returned by get_point_path. If changed, update needs to be called before finding the next path.

Heuristic **default_compute_heuristic** = `0` 🔗

- void **set_default_compute_heuristic**(value: Heuristic)
- Heuristic **get_default_compute_heuristic**()

The default Heuristic which will be used to calculate the cost between two points if _compute_cost was not overridden.

Heuristic **default_estimate_heuristic** = `0` 🔗

- void **set_default_estimate_heuristic**(value: Heuristic)
- Heuristic **get_default_estimate_heuristic**()

The default Heuristic which will be used to calculate the cost between the point and the end point if _estimate_cost was not overridden.

DiagonalMode **diagonal_mode** = `0` 🔗

- void **set_diagonal_mode**(value: DiagonalMode)
- DiagonalMode **get_diagonal_mode**()

A specific DiagonalMode mode which will force the path to avoid or accept the specified diagonals.

---

bool **jumping_enabled** = `false` 🔗

- void **set_jumping_enabled**(value: bool)
- bool **is_jumping_enabled**()

Enables or disables jumping to skip up the intermediate points and speeds up the searching algorithm.

Note: Currently, toggling it on disables the consideration of weight scaling in pathfinding.

Vector2 **offset** = `Vector2(0, 0)`

- void **set_offset**(value: Vector2)
- Vector2 **get_offset**()

The offset of the grid which will be applied to calculate the resulting point position returned by get_point_path. If changed, update needs to be called before finding the next path.

Rect2i **region** = `Rect2i(0, 0, 0, 0)`

- void **set_region**(value: Rect2i)
- Rect2i **get_region**()

The region of grid cells available for pathfinding. If changed, update needs to be called before finding the next path.

Vector2i **size** = `Vector2i(0, 0)`

- void **set_size**(value: Vector2i)
- Vector2i **get_size**()

Deprecated: Use region instead.

The size of the grid (number of cells of size cell_size on each axis). If changed, update needs to be called before finding the next path.

Method Descriptions

float **_compute_cost**(from_id: Vector2i, to_id: Vector2i) virtual const 🔗

Called when computing the cost between two connected points.

Note that this function is hidden in the default **AStarGrid2D** class.

---

float **_estimate_cost**(from_id: Vector2i, to_id: Vector2i) virtual const 🔗

Called when estimating the cost between a point and the path's ending point.

Note that this function is hidden in the default **AStarGrid2D** class.

---

void **clear**() 🔗

Clears the grid and sets the region to `Rect2i(0, 0, 0, 0)`.

---

void **fill_solid_region**(region: Rect2i, solid: bool = true) 🔗

Fills the given `region` on the grid with the specified value for the solid flag.

Note: Calling update is not needed after the call of this function.

---

void **fill_weight_scale_region**(region: Rect2i, weight_scale: float) 🔗

Fills the given `region` on the grid with the specified value for the weight scale.

Note: Calling update is not needed after the call of this function.

Array[Vector2i] **get_id_path**(from_id: Vector2i, to_id: Vector2i, allow_partial_path: bool = false)

Returns an array with the IDs of the points that form the path found by AStar2D between the given points. The array is ordered from the starting point to the ending point of the path.

If there is no valid path to the target, and `allow_partial_path` is `true`, returns a path to the point closest to the target that can be reached.

PackedVector2Array **get_point_path**(from_id: Vector2i, to_id: Vector2i, allow_partial_path: bool = false)

Returns an array with the points that are in the path found by **AStarGrid2D** between the given points. The array is ordered from the starting point to the ending point of the path.

If there is no valid path to the target, and `allow_partial_path` is `true`, returns a path to the point closest to the target that can be reached.

Note: This method is not thread-safe. If called from a Thread, it will return an empty array and will print an error message.

Vector2 **get_point_position**(id: Vector2i) const

Returns the position of the point associated with the given `id`.

---

float **get_point_weight_scale**(id: Vector2i) const

Returns the weight scale of the point associated with the given `id`.

---

bool **is_dirty**() const

Indicates that the grid parameters were changed and update needs to be called.

---

bool **is_in_bounds**(x: int, y: int) const

Returns `true` if the `x` and `y` is a valid grid coordinate (id), i.e. if it is inside region. Equivalent to `region.has_point(Vector2i(x, y))`.

---

bool **is_in_boundsv**(id: Vector2i) const

Returns `true` if the `id` vector is a valid grid coordinate, i.e. if it is inside region. Equivalent to `region.has_point(id)`.

---

bool **is_point_solid**(id: Vector2i) const

Returns `true` if a point is disabled for pathfinding. By default, all points are enabled.

---

void **set_point_solid**(id: Vector2i, solid: bool = true)

Disables or enables the specified point for pathfinding. Useful for making an obstacle. By default, all points are enabled.

Note: Calling update is not needed after the call of this function.

---

void **set_point_weight_scale**(id: Vector2i, weight_scale: float)

Sets the `weight_scale` for the point with the given `id`. The `weight_scale` is multiplied by the result of _compute_cost when determining the overall cost of traveling across a segment from a neighboring point to this point.

Note: Calling update is not needed after the call of this function.

---

void **update**() 🔗

Updates the internal state of the grid according to the parameters to prepare it to search the path. Needs to be called if parameters like region, cell_size or offset are changed. is_dirty will return `true` if this is the case and this needs to be called.

Note: All point data (solidity and weight scale) will be cleared.

AudioEffectInstance

Inherits: RefCounted **<** Object

Inherited By: AudioEffectSpectrumAnalyzerInstance

Manipulates the audio it receives for a given effect.

Description

An audio effect instance manipulates the audio it receives for a given effect. This instance is automatically created by an AudioEffect when it is added to a bus, and should usually not be created directly. If necessary, it can be fetched at run-time with AudioServer.get_bus_effect_instance.

Tutorials

- Audio buses

Methods

void	_process(src_buffer: `const void*`, dst_buffer: `AudioFrame*`, frame_count: int) virtual
bool	_process_silence() virtual const

Method Descriptions

void **_process**(src_buffer: `const void*`, dst_buffer: `AudioFrame*`, frame_count: int) virtual 🔗

Called by the AudioServer to process this effect. When _process_silence is not overridden or it returns `false`, this method is called only when the bus is active.

Note: It is not useful to override this method in GDScript or C#. Only GDExtension can take advantage of it.

---

bool **_process_silence**() virtual const 🔗

Override this method to customize the processing behavior of this effect instance.

Should return `true` to force the AudioServer to always call _process, even if the bus has been muted or cannot otherwise be heard.

AudioEffectSpectrumAnalyzerInstance

Inherits: AudioEffectInstance **<** RefCounted **<** Object

Queryable instance of an AudioEffectSpectrumAnalyzer.

Description

The runtime part of an AudioEffectSpectrumAnalyzer, which can be used to query the magnitude of a frequency range on its host bus.

An instance of this class can be acquired with AudioServer.get_bus_effect_instance.

Tutorials

- Audio Spectrum Visualizer Demo [https://godotengine.org/asset-library/asset/2762]

Methods

Vector2	get_magnitude_for_frequency_range(from_hz: float, to_hz: float, mode: MagnitudeMode = 1) const

Enumerations

enum **MagnitudeMode**:

MagnitudeMode **MAGNITUDE_AVERAGE** = `0`

Use the average value across the frequency range as magnitude.

MagnitudeMode **MAGNITUDE_MAX** = `1`

Use the maximum value of the frequency range as magnitude.

Method Descriptions

Vector2 **get_magnitude_for_frequency_range**(from_hz: float, to_hz: float, mode: MagnitudeMode = 1) const

Returns the magnitude of the frequencies from `from_hz` to `to_hz` in linear energy as a Vector2. The `x` component of the return value represents the left stereo channel, and `y` represents the right channel.

`mode` determines how the frequency range will be processed. See MagnitudeMode.

AudioSample

Experimental: This class may be changed or removed in future versions.

Inherits: RefCounted **<** Object

Base class for audio samples.

Description

Base class for audio samples.

AudioSamplePlayback

Experimental: This class may be changed or removed in future versions.

Inherits: RefCounted **<** Object

Meta class for playing back audio samples.

Description

Meta class for playing back audio samples.

AudioServer

Inherits: Object

Server interface for low-level audio access.

Description

AudioServer is a low-level server interface for audio access. It is in charge of creating sample data (playable audio) as well as its playback via a voice interface.

Tutorials

- Audio buses
- Audio Device Changer Demo [https://godotengine.org/asset-library/asset/2758]
- Audio Microphone Record Demo [https://godotengine.org/asset-library/asset/2760]
- Audio Spectrum Visualizer Demo [https://godotengine.org/asset-library/asset/2762]

Properties

int	bus_count	`1`
String	input_device	`"Default"`
String	output_device	`"Default"`

float	playback_speed_scale	`1.0`

Methods

void	add_bus(at_position: int = -1)
void	add_bus_effect(bus_idx: int, effect: AudioEffect, at_position: int = -1)
AudioBusLayout	generate_bus_layout() const
int	get_bus_channels(bus_idx: int) const
AudioEffect	get_bus_effect(bus_idx: int, effect_idx: int)
int	get_bus_effect_count(bus_idx: int)
AudioEffectInstance	get_bus_effect_instance(bus_idx: int, effect_idx: int, channel: int = 0)
int	get_bus_index(bus_name: StringName) const
String	get_bus_name(bus_idx: int) const
float	get_bus_peak_volume_left_db(bus_idx: int, channel: int) const

float	get_bus_peak_volume_right_db(bus_idx: int, channel: int) const
StringName	get_bus_send(bus_idx: int) const
float	get_bus_volume_db(bus_idx: int) const
PackedStringArray	get_input_device_list()
float	get_mix_rate() const
PackedStringArray	get_output_device_list()
float	get_output_latency() const
SpeakerMode	get_speaker_mode() const
float	get_time_since_last_mix() const
float	get_time_to_next_mix() const
bool	is_bus_bypassing_effects(bus_idx: int) const
bool	is_bus_effect_enabled(bus_idx: int, effect_idx: int) const

bool	is_bus_mute(bus_idx: int) const
bool	is_bus_solo(bus_idx: int) const
bool	is_stream_registered_as_sample(stream: AudioStream)
void	lock()
void	move_bus(index: int, to_index: int)
void	register_stream_as_sample(stream: AudioStream)
void	remove_bus(index: int)
void	remove_bus_effect(bus_idx: int, effect_idx: int)
void	set_bus_bypass_effects(bus_idx: int, enable: bool)
void	set_bus_effect_enabled(bus_idx: int, effect_idx: int, enabled: bool)
void	set_bus_layout(bus_layout: AudioBusLayout)
void	set_bus_mute(bus_idx: int, enable: bool)

void	set_bus_name(bus_idx: int, name: String)
void	set_bus_send(bus_idx: int, send: StringName)
void	set_bus_solo(bus_idx: int, enable: bool)
void	set_bus_volume_db(bus_idx: int, volume_db: float)
void	set_enable_tagging_used_audio_streams(enable: bool)
void	swap_bus_effects(bus_idx: int, effect_idx: int, by_effect_idx: int)
void	unlock()

Signals

bus_layout_changed() 🔗

Emitted when an audio bus is added, deleted, or moved.

bus_renamed(bus_index: int, old_name: StringName, new_name: StringName) 🔗

Emitted when the audio bus at `bus_index` is renamed from `old_name` to `new_name`.

Enumerations

enum **SpeakerMode**: 🔗

SpeakerMode **SPEAKER_MODE_STEREO** = `0`

Two or fewer speakers were detected.

SpeakerMode **SPEAKER_SURROUND_31** = `1`

A 3.1 channel surround setup was detected.

SpeakerMode **SPEAKER_SURROUND_51** = `2`

A 5.1 channel surround setup was detected.

SpeakerMode **SPEAKER_SURROUND_71** = `3`

A 7.1 channel surround setup was detected.

---

enum **PlaybackType**: 🔗

PlaybackType **PLAYBACK_TYPE_DEFAULT** = `0`

Experimental: This constant may be changed or removed in future versions.

The playback will be considered of the type declared at ProjectSettings.audio/general/default_playback_type.

PlaybackType **PLAYBACK_TYPE_STREAM** = `1`

Experimental: This constant may be changed or removed in future versions.

Force the playback to be considered as a stream.

PlaybackType **PLAYBACK_TYPE_SAMPLE** = `2`

Experimental: This constant may be changed or removed in future versions.

Force the playback to be considered as a sample. This can provide lower latency and more stable playback (with less risk of audio crackling), at the cost of having less flexibility.

Note: Only currently supported on the web platform.

Note: AudioEffects are not supported when playback is considered as a sample.

PlaybackType **PLAYBACK_TYPE_MAX** = `3`

Experimental: This constant may be changed or removed in future versions.

Represents the size of the PlaybackType enum.

Property Descriptions

int **bus_count** = `1` 🔗

- void **set_bus_count**(value: int)
- int **get_bus_count**()

Number of available audio buses.

String **input_device** = `"Default"` 🔗

- void **set_input_device**(value: String)
- String **get_input_device**()

Name of the current device for audio input (see get_input_device_list). On systems with multiple audio inputs (such as analog, USB and HDMI audio), this can be used to select the audio input device. The value `"Default"` will record audio on the system-wide default audio input. If an invalid device name is set, the value will be reverted back to `"Default"`.

Note: ProjectSettings.audio/driver/enable_input must be `true` for audio input to work. See also that setting's description for caveats related to permissions and operating system privacy settings.

String **output_device** = `"Default"` 🔗

- void **set_output_device**(value: String)
- String **get_output_device**()

Name of the current device for audio output (see get_output_device_list). On systems with multiple audio outputs (such as analog, USB and HDMI audio), this can be used to select the audio output device. The value `"Default"` will play audio on the system-wide default audio output. If an invalid device name is set, the value will be reverted back to `"Default"`.

---

float **playback_speed_scale** = `1.0` 🔗

- void **set_playback_speed_scale**(value: float)
- float **get_playback_speed_scale**()

Scales the rate at which audio is played (i.e. setting it to `0.5` will make the audio be played at half its speed). See also Engine.time_scale to affect the general simulation speed, which is independent from playback_speed_scale.

Method Descriptions

void **add_bus**(at_position: int = -1)

Adds a bus at `at_position`.

---

void **add_bus_effect**(bus_idx: int, effect: AudioEffect, at_position: int = -1)

Adds an AudioEffect effect to the bus `bus_idx` at `at_position`.

AudioBusLayout **generate_bus_layout**() const

Generates an AudioBusLayout using the available buses and effects.

---

int **get_bus_channels**(bus_idx: int) const

Returns the number of channels of the bus at index `bus_idx`.

AudioEffect **get_bus_effect**(bus_idx: int, effect_idx: int)

Returns the AudioEffect at position `effect_idx` in bus `bus_idx`.

---

int **get_bus_effect_count**(bus_idx: int)

Returns the number of effects on the bus at `bus_idx`.

AudioEffectInstance **get_bus_effect_instance**(bus_idx: int, effect_idx: int, channel: int = 0)

Returns the AudioEffectInstance assigned to the given bus and effect indices (and optionally channel).

---

int **get_bus_index**(bus_name: StringName) const

Returns the index of the bus with the name `bus_name`. Returns `-1` if no bus with the specified name exist.

String **get_bus_name**(bus_idx: int) const

Returns the name of the bus with the index `bus_idx`.

---

float **get_bus_peak_volume_left_db**(bus_idx: int, channel: int) const

Returns the peak volume of the left speaker at bus index `bus_idx` and channel index `channel`.

---

float **get_bus_peak_volume_right_db**(bus_idx: int, channel: int) const

Returns the peak volume of the right speaker at bus index `bus_idx` and channel index `channel`.

StringName **get_bus_send**(bus_idx: int) const

Returns the name of the bus that the bus at index `bus_idx` sends to.

---

float **get_bus_volume_db**(bus_idx: int) const

Returns the volume of the bus at index `bus_idx` in dB.

PackedStringArray **get_input_device_list**()

Returns the names of all audio input devices detected on the system.

Note: ProjectSettings.audio/driver/enable_input must be `true` for audio input to work. See also that setting's description for caveats related to permissions and operating system privacy settings.

---

float **get_mix_rate**() const

Returns the sample rate at the output of the **AudioServer**.

PackedStringArray **get_output_device_list**()

Returns the names of all audio output devices detected on the system.

---

float **get_output_latency**() const

Returns the audio driver's effective output latency. This is based on ProjectSettings.audio/driver/output_latency, but the exact returned value will differ depending on the operating system and audio driver.

Note: This can be expensive; it is not recommended to call get_output_latency every frame.

SpeakerMode **get_speaker_mode**() const

Returns the speaker configuration.

---

float **get_time_since_last_mix**() const

Returns the relative time since the last mix occurred.

---

float **get_time_to_next_mix**() const

Returns the relative time until the next mix occurs.

---

bool **is_bus_bypassing_effects**(bus_idx: int) const 🔗

If `true`, the bus at index `bus_idx` is bypassing effects.

---

bool **is_bus_effect_enabled**(bus_idx: int, effect_idx: int) const 🔗

If `true`, the effect at index `effect_idx` on the bus at index `bus_idx` is enabled.

---

bool **is_bus_mute**(bus_idx: int) const 🔗

If `true`, the bus at index `bus_idx` is muted.

---

bool **is_bus_solo**(bus_idx: int) const 🔗

If `true`, the bus at index `bus_idx` is in solo mode.

---

bool **is_stream_registered_as_sample**(stream: AudioStream) 🔗

Experimental: This method may be changed or removed in future versions.

If `true`, the stream is registered as a sample. The engine will not have to register it before playing the sample.

If `false`, the stream will have to be registered before playing it. To prevent lag spikes, register the stream as sample with register_stream_as_sample.

---

void **lock**() 🔗

Locks the audio driver's main loop.

Note: Remember to unlock it afterwards.

void **move_bus**(index: int, to_index: int)

Moves the bus from index `index` to index `to_index`.

void **register_stream_as_sample**(stream: AudioStream)

Experimental: This method may be changed or removed in future versions.

Forces the registration of a stream as a sample.

Note: Lag spikes may occur when calling this method, especially on single-threaded builds. It is suggested to call this method while loading assets, where the lag spike could be masked, instead of registering the sample right before it needs to be played.

void **remove_bus**(index: int)

Removes the bus at index `index`.

void **remove_bus_effect**(bus_idx: int, effect_idx: int)

Removes the effect at index `effect_idx` from the bus at index `bus_idx`.

void **set_bus_bypass_effects**(bus_idx: int, enable: bool)

If `true`, the bus at index `bus_idx` is bypassing effects.

void **set_bus_effect_enabled**(bus_idx: int, effect_idx: int, enabled: bool)

If `true`, the effect at index `effect_idx` on the bus at index `bus_idx` is enabled.

void **set_bus_layout**(bus_layout: AudioBusLayout)

Overwrites the currently used AudioBusLayout.

void **set_bus_mute**(bus_idx: int, enable: bool)

If `true`, the bus at index `bus_idx` is muted.

void **set_bus_name**(bus_idx: int, name: String)

Sets the name of the bus at index `bus_idx` to `name`.

void **set_bus_send**(bus_idx: int, send: StringName)

Connects the output of the bus at `bus_idx` to the bus named `send`.

void **set_bus_solo**(bus_idx: int, enable: bool)

If `true`, the bus at index `bus_idx` is in solo mode.

void **set_bus_volume_db**(bus_idx: int, volume_db: float)

Sets the volume of the bus at index `bus_idx` to `volume_db`.

void **set_enable_tagging_used_audio_streams**(enable: bool)

If set to `true`, all instances of AudioStreamPlayback will call AudioStreamPlayback._tag_used_streams every mix step.

Note: This is enabled by default in the editor, as it is used by editor plugins for the audio stream previews.

void **swap_bus_effects**(bus_idx: int, effect_idx: int, by_effect_idx: int)

Swaps the position of two effects in bus `bus_idx`.

void **unlock**()

Unlocks the audio driver's main loop. (After locking it, you should always unlock it.)

AudioStreamGeneratorPlayback

Inherits: AudioStreamPlaybackResampled **<** AudioStreamPlayback **<** RefCounted **<** Object

Plays back audio generated using AudioStreamGenerator.

Description

This class is meant to be used with AudioStreamGenerator to play back the generated audio in real-time.

Tutorials

- Audio Generator Demo [https://godotengine.org/asset-library/asset/2759]
- Godot 3.2 will get new audio features [https://godotengine.org/article/godot-32-will-get-new-audio-features]

Methods

bool	can_push_buffer(amount: int) const
void	clear_buffer()
int	get_frames_available() const

int	get_skips() const
bool	push_buffer(frames: PackedVector2Array)
bool	push_frame(frame: Vector2)

Method Descriptions

bool **can_push_buffer**(amount: int) const

Returns `true` if a buffer of the size `amount` can be pushed to the audio sample data buffer without overflowing it, `false` otherwise.

---

void **clear_buffer**()

Clears the audio sample data buffer.

---

int **get_frames_available**() const

Returns the number of frames that can be pushed to the audio sample data buffer without overflowing it. If the result is `0`, the buffer is full.

---

int **get_skips**() const

Returns the number of times the playback skipped due to a buffer underrun in the audio sample data. This value is reset at the start of the playback.

---

bool **push_buffer**(frames: PackedVector2Array) 🔗

Pushes several audio data frames to the buffer. This is usually more efficient than push_frame in C# and compiled languages via GDExtension, but push_buffer may be *less* efficient in GDScript.

---

bool **push_frame**(frame: Vector2) 🔗

Pushes a single audio data frame to the buffer. This is usually less efficient than push_buffer in C# and compiled languages via GDExtension, but push_frame may be *more* efficient in GDScript.

AudioStreamPlayback

Inherits: RefCounted **<** Object

Inherited By: AudioStreamPlaybackInteractive, AudioStreamPlaybackPlaylist, AudioStreamPlaybackPolyphonic, AudioStreamPlaybackResampled, AudioStreamPlaybackSynchronized

Meta class for playing back audio.

Description

Can play, loop, pause a scroll through audio. See AudioStream and AudioStreamOggVorbis for usage.

Tutorials

- Audio Generator Demo [https://godotengine.org/asset-library/asset/2759]

Methods

int	_get_loop_count() virtual const
Variant	_get_parameter(name: StringName) virtual const
float	_get_playback_position() virtual const

bool	_is_playing() virtual const
int	_mix(buffer: `AudioFrame*`, rate_scale: float, frames: int) virtual
void	_seek(position: float) virtual
void	_set_parameter(name: StringName, value: Variant) virtual
void	_start(from_pos: float) virtual
void	_stop() virtual
void	_tag_used_streams() virtual
AudioSamplePlayback	get_sample_playback() const
void	set_sample_playback(playback_sample: AudioSamplePlayback)

Method Descriptions

int **_get_loop_count**() virtual const 🔗

Overridable method. Should return how many times this audio stream has looped. Most built-in playbacks always return `0`.

Variant **_get_parameter**(name: StringName) virtual const

Return the current value of a playback parameter by name (see AudioStream._get_parameter_list).

---

float **_get_playback_position**() virtual const

Overridable method. Should return the current progress along the audio stream, in seconds.

---

bool **_is_playing**() virtual const

Overridable method. Should return `true` if this playback is active and playing its audio stream.

---

int **_mix**(buffer: `AudioFrame*`, rate_scale: float, frames: int) virtual

Override this method to customize how the audio stream is mixed. This method is called even if the playback is not active.

Note: It is not useful to override this method in GDScript or C#. Only GDExtension can take advantage of it.

---

void **_seek**(position: float) virtual

Override this method to customize what happens when seeking this audio stream at the given `position`, such as by calling AudioStreamPlayer.seek.

---

void **_set_parameter**(name: StringName, value: Variant) virtual

Set the current value of a playback parameter by name (see AudioStream._get_parameter_list).

---

void **_start**(from_pos: float) virtual

Override this method to customize what happens when the playback starts at the given position, such as by calling AudioStreamPlayer.play.

---

void **_stop**() virtual

Override this method to customize what happens when the playback is stopped, such as by calling AudioStreamPlayer.stop.

---

void **_tag_used_streams**() virtual

Overridable method. Called whenever the audio stream is mixed if the playback is active and AudioServer.set_enable_tagging_used_audio_streams has been set to `true`. Editor plugins may use this method to "tag" the current position along the audio stream and display it in a preview.

AudioSamplePlayback **get_sample_playback**() const

Experimental: This method may be changed or removed in future versions.

Returns the AudioSamplePlayback associated with this **AudioStreamPlayback** for playing back the audio sample of this stream.

void **set_sample_playback**(playback_sample: AudioSamplePlayback)

Experimental: This method may be changed or removed in future versions.

Associates AudioSamplePlayback to this **AudioStreamPlayback** for playing back the audio sample of this stream.

AudioStreamPlaybackInteractive

Inherits: AudioStreamPlayback **<** RefCounted **<** Object

Playback component of AudioStreamInteractive.

Description

Playback component of AudioStreamInteractive. Contains functions to change the currently played clip.

Methods

void	switch_to_clip(clip_index: int)
void	switch_to_clip_by_name(clip_name: StringName)

Method Descriptions

void **switch_to_clip**(clip_index: int) 🔗

Switch to a clip (by index).

void **switch_to_clip_by_name**(clip_name: StringName) 🔗

Switch to a clip (by name).

AudioStreamPlaybackOggVorbis

Inherits: AudioStreamPlaybackResampled **<** AudioStreamPlayback **<** RefCounted **<** Object

There is currently no description for this class. Please help us by contributing one!

AudioStreamPlaybackPlaylist

Inherits: AudioStreamPlayback **<** RefCounted **<** Object

Playback class used for AudioStreamPlaylist.

AudioStreamPlaybackPolyphonic

Inherits: AudioStreamPlayback < RefCounted < Object

Playback instance for AudioStreamPolyphonic.

Description

Playback instance for AudioStreamPolyphonic. After setting the `stream` property of AudioStreamPlayer, AudioStreamPlayer2D, or AudioStreamPlayer3D, the playback instance can be obtained by calling AudioStreamPlayer.get_stream_playback, AudioStreamPlayer2D.get_stream_playback or AudioStreamPlayer3D.get_stream_playback methods.

Methods

bool	is_stream_playing(stream: int) const
int	play_stream(stream: AudioStream, from_offset: float = 0, volume_db: float = 0, pitch_scale: float = 1.0, playback_type: PlaybackType = 0, bus: StringName = &"Master")
void	set_stream_pitch_scale(stream: int, pitch_scale: float)

void	set_stream_volume(stream: int, volume_db: float)
void	stop_stream(stream: int)

Constants

INVALID_ID = `-1`

Returned by play_stream in case it could not allocate a stream for playback.

Method Descriptions

bool **is_stream_playing**(stream: int) const

Return true whether the stream associated with an integer ID is still playing. Check play_stream for information on when this ID becomes invalid.

---

int **play_stream**(stream: AudioStream, from_offset: float = 0, volume_db: float = 0, pitch_scale: float = 1.0, playback_type: PlaybackType = 0, bus: StringName = &"Master")

Play an AudioStream at a given offset, volume, pitch scale, playback type, and bus. Playback starts immediately.

The return value is a unique integer ID that is associated to this playback stream and which can be used to control it.

This ID becomes invalid when the stream ends (if it does not loop), when the **AudioStreamPlaybackPolyphonic** is stopped, or when stop_stream is called.

This function returns INVALID_ID if the amount of streams currently playing equals AudioStreamPolyphonic.polyphony. If you need a higher amount of maximum polyphony, raise this value.

void **set_stream_pitch_scale**(stream: int, pitch_scale: float)

Change the stream pitch scale. The `stream` argument is an integer ID returned by play_stream.

void **set_stream_volume**(stream: int, volume_db: float)

Change the stream volume (in db). The `stream` argument is an integer ID returned by play_stream.

void **stop_stream**(stream: int)

Stop a stream. The `stream` argument is an integer ID returned by play_stream, which becomes invalid after calling this function.

AudioStreamPlaybackResampled

Inherits: AudioStreamPlayback < RefCounted < Object

Inherited By: AudioStreamGeneratorPlayback, AudioStreamPlaybackOggVorbis

There is currently no description for this class. Please help us by contributing one!

Methods

float	_get_stream_sampling_rate() virtual const
int	_mix_resampled(dst_buffer: `AudioFrame*`, frame_count: int) virtual
void	begin_resample()

Method Descriptions

float **_get_stream_sampling_rate**() virtual const 🔗

There is currently no description for this method. Please help us by contributing one!

---

int **_mix_resampled**(dst_buffer: `AudioFrame*`, frame_count: int) virtual

There is currently no description for this method. Please help us by contributing one!

---

void **begin_resample**()

There is currently no description for this method. Please help us by contributing one!

AudioStreamPlaybackSynchronized

Inherits: AudioStreamPlayback **<** RefCounted **<** Object

There is currently no description for this class. Please help us by contributing one!

CallbackTweener

Inherits: Tweener **<** RefCounted **<** Object

Calls the specified method after optional delay.

Description

CallbackTweener is used to call a method in a tweening sequence. See Tween.tween_callback for more usage information.

The tweener will finish automatically if the callback's target object is freed.

Note: Tween.tween_callback is the only correct way to create **CallbackTweener**. Any **CallbackTweener** created manually will not function correctly.

Methods

CallbackTweener	set_delay(delay: float)

Method Descriptions

CallbackTweener **set_delay**(delay: float) 🔗

Makes the callback call delayed by given time in seconds.

Example:

```
var tween = get_tree().create_tween()
tween.tween_callback(queue_free).set_delay(2) #this will call queue_free() after 2 seconds
```

CameraFeed

Inherits: RefCounted **<** Object

A camera feed gives you access to a single physical camera attached to your device.

Description

A camera feed gives you access to a single physical camera attached to your device. When enabled, Godot will start capturing frames from the camera which can then be used. See also CameraServer.

Note: Many cameras will return YCbCr images which are split into two textures and need to be combined in a shader. Godot does this automatically for you if you set the environment to show the camera image in the background.

Properties

bool	feed_is_active	`false`
Transform2D	feed_transform	`Transform2D(1, 0, 0, -1, 0, 1)`

Methods

FeedDataType	get_datatype() const

int	get_id() const
String	get_name() const
FeedPosition	get_position() const

Enumerations

enum **FeedDataType**: 🔗

FeedDataType **FEED_NOIMAGE** = `0`

No image set for the feed.

FeedDataType **FEED_RGB** = `1`

Feed supplies RGB images.

FeedDataType **FEED_YCBCR** = `2`

Feed supplies YCbCr images that need to be converted to RGB.

FeedDataType **FEED_YCBCR_SEP** = `3`

Feed supplies separate Y and CbCr images that need to be combined and converted to RGB.

---

enum **FeedPosition**: 🔗

FeedPosition **FEED_UNSPECIFIED** = `0`

Unspecified position.

FeedPosition **FEED_FRONT** = `1`

Camera is mounted at the front of the device.

FeedPosition **FEED_BACK** = `2`

Camera is mounted at the back of the device.

Property Descriptions

bool **feed_is_active** = `false`

- void **set_active**(value: bool)
- bool **is_active**()

If `true`, the feed is active.

Transform2D **feed_transform** = `Transform2D(1, 0, 0, -1, 0, 1)`

- void **set_transform**(value: Transform2D)
- Transform2D **get_transform**()

The transform applied to the camera's image.

Method Descriptions

FeedDataType **get_datatype**() const

Returns feed image data type.

---

int **get_id**() const 🔗

Returns the unique ID for this feed.

String **get_name**() const 🔗

Returns the camera's name.

FeedPosition **get_position**() const 🔗

Returns the position of camera on the device.

CameraServer

Inherits: Object

Server keeping track of different cameras accessible in Godot.

Description

The **CameraServer** keeps track of different cameras accessible in Godot. These are external cameras such as webcams or the cameras on your phone.

It is notably used to provide AR modules with a video feed from the camera.

Note: This class is currently only implemented on macOS and iOS. To get a CameraFeed on iOS, the camera plugin from godot-ios-plugins [https://github.com/godotengine/godot-ios-plugins] is required. On other platforms, no CameraFeeds will be available.

Methods

void	add_feed(feed: CameraFeed)
Array[CameraFeed]	feeds()
CameraFeed	get_feed(index: int)
int	get_feed_count()

void	remove_feed(feed: CameraFeed)

Signals

camera_feed_added(id: int)

Emitted when a CameraFeed is added (e.g. a webcam is plugged in).

camera_feed_removed(id: int)

Emitted when a CameraFeed is removed (e.g. a webcam is unplugged).

Enumerations

enum **FeedImage**:

FeedImage **FEED_RGBA_IMAGE** = `0`

The RGBA camera image.

FeedImage **FEED_YCBCR_IMAGE** = `0`

The YCbCr [https://en.wikipedia.org/wiki/YCbCr] camera image.

FeedImage **FEED_Y_IMAGE** = `0`

The Y component camera image.

FeedImage **FEED_CBCR_IMAGE** = `1`

The CbCr component camera image.

Method Descriptions

void **add_feed**(feed: CameraFeed)

Adds the camera `feed` to the camera server.

Array[CameraFeed] **feeds**()

Returns an array of CameraFeeds.

CameraFeed **get_feed**(index: int)

Returns the CameraFeed corresponding to the camera with the given `index`.

int **get_feed_count**()

Returns the number of CameraFeeds registered.

void **remove_feed**(feed: CameraFeed)

Removes the specified camera `feed`.

CharFXTransform

Inherits: RefCounted **<** Object

Controls how an individual character will be displayed in a RichTextEffect.

Description

By setting various properties on this object, you can control how individual characters will be displayed in a RichTextEffect.

Tutorials

- BBCode in RichTextLabel
- RichTextEffect test project (third-party) [https://github.com/Eoin-ONeill-Yokai/Godot-Rich-Text-Effect-Test-Project]

Properties

Color	color	`Color(0, 0, 0, 1)`
float	elapsed_time	`0.0`
Dictionary	env	`{}`
RID	font	`RID()`

int	glyph_count	`0`
int	glyph_flags	`0`
int	glyph_index	`0`
Vector2	offset	`Vector2(0, 0)`
bool	outline	`false`
Vector2i	range	`Vector2i(0, 0)`
int	relative_index	`0`
Transform2D	transform	`Transform2D(1, 0, 0, 1, 0, 0)`
bool	visible	`true`

Property Descriptions

Color **color** = `Color(0, 0, 0, 1)` 🔗

- void **set_color**(value: Color)
- Color **get_color**()

The color the character will be drawn with.

float **elapsed_time** = `0.0`

- void **set_elapsed_time**(value: float)
- float **get_elapsed_time**()

The time elapsed since the RichTextLabel was added to the scene tree (in seconds). Time stops when the RichTextLabel is paused (see Node.process_mode). Resets when the text in the RichTextLabel is changed.

Note: Time still passes while the RichTextLabel is hidden.

Dictionary **env** = `{}`

- void **set_environment**(value: Dictionary)
- Dictionary **get_environment**()

Contains the arguments passed in the opening BBCode tag. By default, arguments are strings; if their contents match a type such as bool, int or float, they will be converted automatically. Color codes in the form `#rrggbb` or `#rgb` will be converted to an opaque Color. String arguments may not contain spaces, even if they're quoted. If present, quotes will also be present in the final string.

For example, the opening BBCode tag `[example foo=hello bar=true baz=42 color=#ffffff]` will map to the following Dictionary:

```
{"foo": "hello", "bar": true, "baz": 42, "color": Color(1, 1,
1, 1)}
```

RID **font** = `RID()`

- void **set_font**(value: RID)
- RID **get_font**()

Font resource used to render glyph.

int **glyph_count** = `0`

- void **set_glyph_count**(value: int)
- int **get_glyph_count**()

Number of glyphs in the grapheme cluster. This value is set in the first glyph of a cluster. Setting this property won't affect drawing.

int **glyph_flags** = `0`

- void **set_glyph_flags**(value: int)
- int **get_glyph_flags**()

Glyph flags. See GraphemeFlag for more info. Setting this property won't affect drawing.

int **glyph_index** = `0`

- void **set_glyph_index**(value: int)
- int **get_glyph_index**()

Font specific glyph index.

Vector2 **offset** = `Vector2(0, 0)`

- void **set_offset**(value: Vector2)
- Vector2 **get_offset**()

The position offset the character will be drawn with (in pixels).

---

bool **outline** = `false`

- void **set_outline**(value: bool)
- bool **is_outline**()

If `true`, FX transform is called for outline drawing. Setting this property won't affect drawing.

Vector2i **range** = `Vector2i(0, 0)`

- void **set_range**(value: Vector2i)
- Vector2i **get_range**()

Absolute character range in the string, corresponding to the glyph. Setting this property won't affect drawing.

---

int **relative_index** = `0`

- void **set_relative_index**(value: int)
- int **get_relative_index**()

The character offset of the glyph, relative to the current RichTextEffect custom block. Setting this property won't affect drawing.

Transform2D **transform** = `Transform2D(1, 0, 0, 1, 0, 0)`

- void **set_transform**(value: Transform2D)
- Transform2D **get_transform**()

The current transform of the current glyph. It can be overridden (for example, by driving the position and rotation from a curve). You can also alter the existing value to apply transforms on top of other effects.

---

bool **visible** = `true`

- void **set_visibility**(value: bool)
- bool **is_visible**()

If `true`, the character will be drawn. If `false`, the character will be hidden. Characters around hidden characters will reflow to take the space of hidden characters. If this is not desired, set their color to `Color(1, 1, 1, 0)` instead.

ClassDB

Inherits: Object

A class information repository.

Description

Provides access to metadata stored for every available class.

Methods

bool	can_instantiate(class: StringName) const
bool	class_exists(class: StringName) const
PackedStringArray	class_get_enum_constants(class: StringName, enum: StringName, no_inheritance: bool = false) const
PackedStringArray	class_get_enum_list(class: StringName, no_inheritance: bool = false) const
int	class_get_integer_constant(class: StringName, name: StringName) const

StringName	class_get_integer_constant_enum(class: StringName, name: StringName, no_inheritance: bool = false) const
PackedStringArray	class_get_integer_constant_list(class: StringName, no_inheritance: bool = false) const
int	class_get_method_argument_count(class: StringName, method: StringName, no_inheritance: bool = false) const
Array[Dictionary]	class_get_method_list(class: StringName, no_inheritance: bool = false) const
Variant	class_get_property(object: Object, property: StringName) const
Variant	class_get_property_default_value(class: StringName, property: StringName) const
Array[Dictionary]	class_get_property_list(class: StringName, no_inheritance: bool = false) const
Dictionary	class_get_signal(class: StringName, signal: StringName) const
Array[Dictionary]	class_get_signal_list(class: StringName, no_inheritance: bool = false) const

bool	class_has_enum(class: StringName, name: StringName, no_inheritance: bool = false) const
bool	class_has_integer_constant(class: StringName, name: StringName) const
bool	class_has_method(class: StringName, method: StringName, no_inheritance: bool = false) const
bool	class_has_signal(class: StringName, signal: StringName) const
Error	class_set_property(object: Object, property: StringName, value: Variant) const
PackedStringArray	get_class_list() const
PackedStringArray	get_inheriters_from_class(class: StringName) const
StringName	get_parent_class(class: StringName) const
Variant	instantiate(class: StringName) const
bool	is_class_enabled(class: StringName) const

bool	is_class_enum_bitfield(class: StringName, enum: StringName, no_inheritance: bool = false) const
bool	is_parent_class(class: StringName, inherits: StringName) const

Method Descriptions

bool **can_instantiate**(class: StringName) const 🔗

Returns `true` if objects can be instantiated from the specified `class`, otherwise returns `false`.

bool **class_exists**(class: StringName) const 🔗

Returns whether the specified `class` is available or not.

PackedStringArray **class_get_enum_constants**(class: StringName, enum: StringName, no_inheritance: bool = false) const 🔗

Returns an array with all the keys in `enum` of `class` or its ancestry.

PackedStringArray **class_get_enum_list**(class: StringName, no_inheritance: bool = false) const 🔗

Returns an array with all the enums of `class` or its ancestry.

int **class_get_integer_constant**(class: StringName, name: StringName) const 🔗

Returns the value of the integer constant `name` of `class` or its ancestry. Always returns 0 when the constant could not be found.

StringName **class_get_integer_constant_enum**(class: StringName, name: StringName, no_inheritance: bool = false) const 🔗

Returns which enum the integer constant `name` of `class` or its ancestry belongs to.

PackedStringArray **class_get_integer_constant_list**(class: StringName, no_inheritance: bool = false) const 🔗

Returns an array with the names all the integer constants of `class` or its ancestry.

int **class_get_method_argument_count**(class: StringName, method: StringName, no_inheritance: bool = false) const 🔗

Returns the number of arguments of the method `method` of `class` or its ancestry if `no_inheritance` is `false`.

Array[Dictionary] **class_get_method_list**(class: StringName, no_inheritance: bool = false) const 🔗

Returns an array with all the methods of `class` or its ancestry if `no_inheritance` is `false`. Every element of the array is a Dictionary with the following keys: `args`, `default_args`, `flags`, `id`, `name`, `return: (class_name, hint, hint_string, name, type, usage)`.

Note: In exported release builds the debug info is not available, so the returned dictionaries will contain only method names.

Variant **class_get_property**(object: Object, property: StringName) const

Returns the value of `property` of `object` or its ancestry.

Variant **class_get_property_default_value**(class: StringName, property: StringName) const

Returns the default value of `property` of `class` or its ancestor classes.

Array[Dictionary] **class_get_property_list**(class: StringName, no_inheritance: bool = false) const

Returns an array with all the properties of `class` or its ancestry if `no_inheritance` is `false`.

Dictionary **class_get_signal**(class: StringName, signal: StringName) const

Returns the `signal` data of `class` or its ancestry. The returned value is a Dictionary with the following keys: `args`, `default_args`, `flags`, `id`, `name`, `return: (class_name, hint, hint_string, name, type, usage)`.

Array[Dictionary] **class_get_signal_list**(class: StringName, no_inheritance: bool = false) const

Returns an array with all the signals of `class` or its ancestry if `no_inheritance` is `false`. Every element of the array is a Dictionary as described in class_get_signal.

---

bool **class_has_enum**(class: StringName, name: StringName, no_inheritance: bool = false) const

Returns whether `class` or its ancestry has an enum called `name` or not.

---

bool **class_has_integer_constant**(class: StringName, name: StringName) const

Returns whether `class` or its ancestry has an integer constant called `name` or not.

---

bool **class_has_method**(class: StringName, method: StringName, no_inheritance: bool = false) const

Returns whether `class` (or its ancestry if `no_inheritance` is `false`) has a method called `method` or not.

---

bool **class_has_signal**(class: StringName, signal: StringName) const

Returns whether `class` or its ancestry has a signal called `signal` or not.

Error **class_set_property**(object: Object, property: StringName, value: Variant) const

Sets `property` value of `object` to `value`.

PackedStringArray **get_class_list**() const 🔗

Returns the names of all the classes available.

PackedStringArray **get_inheriters_from_class**(class: StringName) const 🔗

Returns the names of all the classes that directly or indirectly inherit from `class`.

StringName **get_parent_class**(class: StringName) const 🔗

Returns the parent class of `class`.

Variant **instantiate**(class: StringName) const 🔗

Creates an instance of `class`.

---

bool **is_class_enabled**(class: StringName) const 🔗

Returns whether this `class` is enabled or not.

---

bool **is_class_enum_bitfield**(class: StringName, enum: StringName, no_inheritance: bool = false) const 🔗

Returns whether `class` (or its ancestor classes if `no_inheritance` is `false`) has an enum called `enum` that is a bitfield.

---

bool **is_parent_class**(class: StringName, inherits: StringName) const 🔗

Returns whether `inherits` is an ancestor of `class` or not.

ConfigFile

Inherits: RefCounted < Object

Helper class to handle INI-style files.

Description

This helper class can be used to store Variant values on the filesystem using INI-style formatting. The stored values are identified by a section and a key:

```
[section]
some_key=42
string_example="Hello World3D!"
a_vector=Vector3(1, 0, 2)
```

The stored data can be saved to or parsed from a file, though ConfigFile objects can also be used directly without accessing the filesystem.

The following example shows how to create a simple **ConfigFile** and save it on disc:

GDScript

```
# Create new ConfigFile object.
var config = ConfigFile.new()

# Store some values.
config.set_value("Player1", "player_name", "Steve")
config.set_value("Player1", "best_score", 10)
config.set_value("Player2", "player_name", "V3geta")
config.set_value("Player2", "best_score", 9001)
```

```
# Save it to a file (overwrite if already exists).
config.save("user://scores.cfg")
```

C#

```
// Create new ConfigFile object.
var config = new ConfigFile();

// Store some values.
config.SetValue("Player1", "player_name", "Steve");
config.SetValue("Player1", "best_score", 10);
config.SetValue("Player2", "player_name", "V3geta");
config.SetValue("Player2", "best_score", 9001);

// Save it to a file (overwrite if already exists).
config.Save("user://scores.cfg");
```

This example shows how the above file could be loaded:

GDScript

```
var score_data = {}
var config = ConfigFile.new()

# Load data from a file.
var err = config.load("user://scores.cfg")

# If the file didn't load, ignore it.
if err != OK:
    return

# Iterate over all sections.
for player in config.get_sections():
    # Fetch the data for each section.
    var player_name = config.get_value(player, "player_name")
    var player_score = config.get_value(player, "best_score")
    score_data[player_name] = player_score
```

C#

```
var score_data = new Godot.Collections.Dictionary();
var config = new ConfigFile();

// Load data from a file.
Error err = config.Load("user://scores.cfg");

// If the file didn't load, ignore it.
if (err != Error.Ok)
{
    return;
}

// Iterate over all sections.
foreach (String player in config.GetSections())
{
    // Fetch the data for each section.
    var player_name = (String)config.GetValue(player,
"player_name");
    var player_score = (int)config.GetValue(player,
"best_score");
    score_data[player_name] = player_score;
}
```

Any operation that mutates the ConfigFile such as set_value, clear, or erase_section, only changes what is loaded in memory. If you want to write the change to a file, you have to save the changes with save, save_encrypted, or save_encrypted_pass.

Keep in mind that section and property names can't contain spaces. Anything after a space will be ignored on save and on load.

ConfigFiles can also contain manually written comment lines starting with a semicolon (`;`). Those lines will be ignored when parsing the file. Note that comments will be lost when saving the ConfigFile. This can still be useful for dedicated server configuration files, which are typically never overwritten without explicit user action.

Note: The file extension given to a ConfigFile does not have any impact on its formatting or behavior. By convention, the `.cfg` extension is used here, but any other extension such as `.ini` is also valid. Since neither `.cfg` nor `.ini` are standardized, Godot's ConfigFile formatting may differ from files written by other programs.

Methods

void	clear()
String	encode_to_text() const
void	erase_section(section: String)
void	erase_section_key(section: String, key: String)
PackedStringArray	get_section_keys(section: String) const
PackedStringArray	get_sections() const
Variant	get_value(section: String, key: String, default: Variant = null) const
bool	has_section(section: String) const
bool	has_section_key(section: String, key: String) const

Error	load(path: String)
Error	load_encrypted(path: String, key: PackedByteArray)
Error	load_encrypted_pass(path: String, password: String)
Error	parse(data: String)
Error	save(path: String)
Error	save_encrypted(path: String, key: PackedByteArray)
Error	save_encrypted_pass(path: String, password: String)
void	set_value(section: String, key: String, value: Variant)

Method Descriptions

void **clear**() 🔗

Removes the entire contents of the config.

String **encode_to_text**() const

Obtain the text version of this config file (the same text that would be written to a file).

---

void **erase_section**(section: String)

Deletes the specified section along with all the key-value pairs inside. Raises an error if the section does not exist.

---

void **erase_section_key**(section: String, key: String)

Deletes the specified key in a section. Raises an error if either the section or the key do not exist.

PackedStringArray **get_section_keys**(section: String) const

Returns an array of all defined key identifiers in the specified section. Raises an error and returns an empty array if the section does not exist.

PackedStringArray **get_sections**() const

Returns an array of all defined section identifiers.

Variant **get_value**(section: String, key: String, default: Variant = null) const

Returns the current value for the specified section and key. If either the section or the key do not exist, the method returns the fallback `default` value. If `default` is not specified or set to `null`, an error is also raised.

---

bool **has_section**(section: String) const 🔗

Returns `true` if the specified section exists.

---

bool **has_section_key**(section: String, key: String) const 🔗

Returns `true` if the specified section-key pair exists.

Error **load**(path: String) 🔗

Loads the config file specified as a parameter. The file's contents are parsed and loaded in the **ConfigFile** object which the method was called on.

Returns @GlobalScope.OK on success, or one of the other Error values if the operation failed.

Error **load_encrypted**(path: String, key: PackedByteArray) 🔗

Loads the encrypted config file specified as a parameter, using the provided `key` to decrypt it. The file's contents are parsed and loaded in the **ConfigFile** object which the method was called on.

Returns @GlobalScope.OK on success, or one of the other Error values if the operation failed.

Error **load_encrypted_pass**(path: String, password: String) 🔗

Loads the encrypted config file specified as a parameter, using the provided `password` to decrypt it. The file's contents are parsed and loaded in the **ConfigFile** object which the method was called on.

Returns @GlobalScope.OK on success, or one of the other Error values if the operation failed.

Error **parse**(data: String) 🔗

Parses the passed string as the contents of a config file. The string is parsed and loaded in the ConfigFile object which the method was called on.

Returns @GlobalScope.OK on success, or one of the other Error values if the operation failed.

Error **save**(path: String) 🔗

Saves the contents of the **ConfigFile** object to the file specified as a parameter. The output file uses an INI-style structure.

Returns @GlobalScope.OK on success, or one of the other Error values if the operation failed.

Error **save_encrypted**(path: String, key: PackedByteArray) 🔗

Saves the contents of the **ConfigFile** object to the AES-256 encrypted file specified as a parameter, using the provided `key` to encrypt it. The output file uses an INI-style structure.

Returns @GlobalScope.OK on success, or one of the other Error values if the operation failed.

Error **save_encrypted_pass**(path: String, password: String) 🔗

Saves the contents of the **ConfigFile** object to the AES-256 encrypted file specified as a parameter, using the provided `password` to encrypt it. The output file uses an INI-style structure.

Returns @GlobalScope.OK on success, or one of the other Error values if the operation failed.

---

void **set_value**(section: String, key: String, value: Variant) 🔗

Assigns a value to the specified key of the specified section. If either the section or the key do not exist, they are created. Passing a `null` value deletes the specified key if it exists, and deletes the section if it ends up empty once the key has been removed.

Crypto

Inherits: RefCounted **<** Object

Provides access to advanced cryptographic functionalities.

Description

The Crypto class provides access to advanced cryptographic functionalities.

Currently, this includes asymmetric key encryption/decryption, signing/verification, and generating cryptographically secure random bytes, RSA keys, HMAC digests, and self-signed X509Certificates.

GDScript

```
var crypto = Crypto.new()

# Generate new RSA key.
var key = crypto.generate_rsa(4096)

# Generate new self-signed certificate with the given key.
var cert = crypto.generate_self_signed_certificate(key,
"CN=mydomain.com,O=My Game Company,C=IT")

# Save key and certificate in the user folder.
key.save("user://generated.key")
cert.save("user://generated.crt")

# Encryption
var data = "Some data"
var encrypted = crypto.encrypt(key, data.to_utf8_buffer())

# Decryption
var decrypted = crypto.decrypt(key, encrypted)
```

```
# Signing
var signature = crypto.sign(HashingContext.HASH_SHA256,
data.sha256_buffer(), key)

# Verifying
var verified = crypto.verify(HashingContext.HASH_SHA256,
data.sha256_buffer(), signature, key)

# Checks
assert(verified)
assert(data.to_utf8_buffer() == decrypted)
```

C#

```
using Godot;
using System.Diagnostics;

Crypto crypto = new Crypto();

// Generate new RSA key.
CryptoKey key = crypto.GenerateRsa(4096);

// Generate new self-signed certificate with the given key.
X509Certificate cert =
crypto.GenerateSelfSignedCertificate(key, "CN=mydomain.com,O=My
Game Company,C=IT");

// Save key and certificate in the user folder.
key.Save("user://generated.key");
cert.Save("user://generated.crt");

// Encryption
string data = "Some data";
byte[] encrypted = crypto.Encrypt(key, data.ToUtf8Buffer());

// Decryption
byte[] decrypted = crypto.Decrypt(key, encrypted);

// Signing
byte[] signature = crypto.Sign(HashingContext.HashType.Sha256,
Data.Sha256Buffer(), key);
```

```
// Verifying
bool verified = crypto.Verify(HashingContext.HashType.Sha256,
Data.Sha256Buffer(), signature, key);

// Checks
Debug.Assert(verified);
Debug.Assert(data.ToUtf8Buffer() == decrypted);
```

Methods

bool	constant_time_compare(trusted: PackedByteArray, received: PackedByteArray)
PackedByteArray	decrypt(key: CryptoKey, ciphertext: PackedByteArray)
PackedByteArray	encrypt(key: CryptoKey, plaintext: PackedByteArray)
PackedByteArray	generate_random_bytes(size: int)
CryptoKey	generate_rsa(size: int)
X509Certificate	generate_self_signed_certificate(key: CryptoKey, issuer_name: String = "CN=myserver,O=myorganisation,C=IT", not_before: String = "20140101000000", not_after: String = "20340101000000")

PackedByteArray	hmac_digest(hash_type: HashType, key: PackedByteArray, msg: PackedByteArray)
PackedByteArray	sign(hash_type: HashType, hash: PackedByteArray, key: CryptoKey)
bool	verify(hash_type: HashType, hash: PackedByteArray, signature: PackedByteArray, key: CryptoKey)

Method Descriptions

bool **constant_time_compare**(trusted: PackedByteArray, received: PackedByteArray) 🔗

Compares two PackedByteArrays for equality without leaking timing information in order to prevent timing attacks.

See this blog post [https://paragonie.com/blog/2015/11/preventing-timing-attacks-on-string-comparison-with-double-hmac-strategy] for more information.

PackedByteArray **decrypt**(key: CryptoKey, ciphertext: PackedByteArray) 🔗

Decrypt the given `ciphertext` with the provided private `key`.

Note: The maximum size of accepted ciphertext is limited by the key size.

PackedByteArray **encrypt**(key: CryptoKey, plaintext: PackedByteArray)

Encrypt the given `plaintext` with the provided public `key`.

Note: The maximum size of accepted plaintext is limited by the key size.

PackedByteArray **generate_random_bytes**(size: int)

Generates a PackedByteArray of cryptographically secure random bytes with given `size`.

CryptoKey **generate_rsa**(size: int)

Generates an RSA CryptoKey that can be used for creating self-signed certificates and passed to StreamPeerTLS.accept_stream.

X509Certificate **generate_self_signed_certificate**(key: CryptoKey, issuer_name: String = "CN=myserver,O=myorganisation,C=IT", not_before: String = "20140101000000", not_after: String = "20340101000000")

Generates a self-signed X509Certificate from the given CryptoKey and `issuer_name`. The certificate validity will be defined by `not_before` and `not_after` (first valid date and last valid date). The `issuer_name` must contain at least "CN=" (common name, i.e. the domain name), "O=" (organization, i.e. your company name), "C=" (country, i.e. 2 lettered ISO-3166 code of the country the organization is based in).

A small example to generate an RSA key and an X509 self-signed certificate.

GDScript

```
var crypto = Crypto.new()
# Generate 4096 bits RSA key.
var key = crypto.generate_rsa(4096)
# Generate self-signed certificate using the given key.
var cert = crypto.generate_self_signed_certificate(key,
"CN=example.com,O=A Game Company,C=IT")
```

C#

```
var crypto = new Crypto();
// Generate 4096 bits RSA key.
CryptoKey key = crypto.GenerateRsa(4096);
// Generate self-signed certificate using the given key.
X509Certificate cert =
crypto.GenerateSelfSignedCertificate(key, "CN=mydomain.com,O=My
Game Company,C=IT");
```

PackedByteArray **hmac_digest**(hash_type: HashType, key: PackedByteArray, msg: PackedByteArray) 🔗

Generates an HMAC [https://en.wikipedia.org/wiki/HMAC] digest of `msg` using `key`. The `hash_type` parameter is the hashing algorithm that is used for the inner and outer hashes.

Currently, only HashingContext.HASH_SHA256 and HashingContext.HASH_SHA1 are supported.

PackedByteArray **sign**(hash_type: HashType, hash: PackedByteArray, key: CryptoKey) 🔗

Sign a given `hash` of type `hash_type` with the provided private `key`.

---

bool **verify**(hash_type: HashType, hash: PackedByteArray, signature: PackedByteArray, key: CryptoKey) 🔗

Verify that a given `signature` for `hash` of type `hash_type` against the provided public `key`.

DirAccess

Inherits: RefCounted **<** Object

Provides methods for managing directories and their content.

Description

This class is used to manage directories and their content, even outside of the project folder.

DirAccess can't be instantiated directly. Instead it is created with a static method that takes a path for which it will be opened.

Most of the methods have a static alternative that can be used without creating a **DirAccess**. Static methods only support absolute paths (including `res://` and `user://`).

```
# Standard
var dir = DirAccess.open("user://levels")
dir.make_dir("world1")
# Static
DirAccess.make_dir_absolute("user://levels/world1")
```

Note: Many resources types are imported (e.g. textures or sound files), and their source asset will not be included in the exported game, as only the imported version is used. Use ResourceLoader to access imported resources.

Here is an example on how to iterate through the files of a directory:

GDScript

```
func dir_contents(path):
    var dir = DirAccess.open(path)
    if dir:
        dir.list_dir_begin()
        var file_name = dir.get_next()
        while file_name != "":
            if dir.current_is_dir():
                print("Found directory: " + file_name)
            else:
                print("Found file: " + file_name)
            file_name = dir.get_next()
    else:
        print("An error occurred when trying to access the
path.")
```

C#

```
public void DirContents(string path)
{
    using var dir = DirAccess.Open(path);
    if (dir != null)
    {
        dir.ListDirBegin();
        string fileName = dir.GetNext();
        while (fileName != "")
        {
            if (dir.CurrentIsDir())
            {
                GD.Print($"Found directory: {fileName}");
            }
            else
            {
                GD.Print($"Found file: {fileName}");
            }
            fileName = dir.GetNext();
        }
    }
    else
    {
        GD.Print("An error occurred when trying to access the
path.");
    }
}
```

Tutorials

- File system

Properties

bool	include_hidden
bool	include_navigational

Methods

Error	change_dir(to_dir: String)
Error	copy(from: String, to: String, chmod_flags: int = -1)
Error	copy_absolute(from: String, to: String, chmod_flags: int = -1) static
Error	create_link(source: String, target: String)
bool	current_is_dir() const
bool	dir_exists(path: String)

bool	dir_exists_absolute(path: String) static
bool	file_exists(path: String)
String	get_current_dir(include_drive: bool = true) const
int	get_current_drive()
PackedStringArray	get_directories()
PackedStringArray	get_directories_at(path: String) static
int	get_drive_count() static
String	get_drive_name(idx: int) static
PackedStringArray	get_files()
PackedStringArray	get_files_at(path: String) static
String	get_next()
Error	get_open_error() static

int	get_space_left()
bool	is_case_sensitive(path: String) const
bool	is_link(path: String)
Error	list_dir_begin()
void	list_dir_end()
Error	make_dir(path: String)
Error	make_dir_absolute(path: String) static
Error	make_dir_recursive(path: String)
Error	make_dir_recursive_absolute(path: String) static
DirAccess	open(path: String) static
String	read_link(path: String)
Error	remove(path: String)

Error	remove_absolute(path: String) static
Error	rename(from: String, to: String)
Error	rename_absolute(from: String, to: String) static

Property Descriptions

bool **include_hidden** 🔗

- void **set_include_hidden**(value: bool)
- bool **get_include_hidden**()

If `true`, hidden files are included when navigating the directory.

Affects list_dir_begin, get_directories and get_files.

bool **include_navigational** 🔗

- void **set_include_navigational**(value: bool)
- bool **get_include_navigational**()

If `true`, `.` and `..` are included when navigating the directory.

Affects list_dir_begin and get_directories.

Method Descriptions

Error **change_dir**(to_dir: String)

Changes the currently opened directory to the one passed as an argument. The argument can be relative to the current directory (e.g. `newdir` or `../newdir`), or an absolute path (e.g. `/tmp/newdir` or `res://somedir/newdir`).

Returns one of the Error code constants (@GlobalScope.OK on success).

Note: The new directory must be within the same scope, e.g. when you had opened a directory inside `res://`, you can't change it to `user://` directory. If you need to open a directory in another access scope, use open to create a new instance instead.

Error **copy**(from: String, to: String, chmod_flags: int = -1)

Copies the `from` file to the `to` destination. Both arguments should be paths to files, either relative or absolute. If the destination file exists and is not access-protected, it will be overwritten.

If `chmod_flags` is different than `-1`, the Unix permissions for the destination path will be set to the provided value, if available on the current operating system.

Returns one of the Error code constants (@GlobalScope.OK on success).

Error **copy_absolute**(from: String, to: String, chmod_flags: int = -1) static

Static version of copy. Supports only absolute paths.

Error **create_link**(source: String, target: String)

Creates symbolic link between files or folders.

Note: On Windows, this method works only if the application is running with elevated privileges or Developer Mode is enabled.

Note: This method is implemented on macOS, Linux, and Windows.

bool **current_is_dir**() const

Returns whether the current item processed with the last get_next call is a directory (`.` and `..` are considered directories).

bool **dir_exists**(path: String)

Returns whether the target directory exists. The argument can be relative to the current directory, or an absolute path.

bool **dir_exists_absolute**(path: String) static

Static version of dir_exists. Supports only absolute paths.

bool **file_exists**(path: String)

Returns whether the target file exists. The argument can be relative to the current directory, or an absolute path.

For a static equivalent, use FileAccess.file_exists.

String **get_current_dir**(include_drive: bool = true) const

Returns the absolute path to the currently opened directory (e.g. `res://folder` or `C:\tmp\folder`).

---

int **get_current_drive**() 🔗

Returns the currently opened directory's drive index. See get_drive_name to convert returned index to the name of the drive.

PackedStringArray **get_directories**() 🔗

Returns a PackedStringArray containing filenames of the directory contents, excluding files. The array is sorted alphabetically.

Affected by include_hidden and include_navigational.

PackedStringArray **get_directories_at**(path: String) static 🔗

Returns a PackedStringArray containing filenames of the directory contents, excluding files, at the given `path`. The array is sorted alphabetically.

Use get_directories if you want more control of what gets included.

---

int **get_drive_count**() static 🔗

On Windows, returns the number of drives (partitions) mounted on the current filesystem.

On macOS, returns the number of mounted volumes.

On Linux, returns the number of mounted volumes and GTK 3 bookmarks.

On other platforms, the method returns 0.

String **get_drive_name**(idx: int) static 🔗

On Windows, returns the name of the drive (partition) passed as an argument (e.g. `C:`).

On macOS, returns the path to the mounted volume passed as an argument.

On Linux, returns the path to the mounted volume or GTK 3 bookmark passed as an argument.

On other platforms, or if the requested drive does not exist, the method returns an empty String.

PackedStringArray **get_files**() 🔗

Returns a PackedStringArray containing filenames of the directory contents, excluding directories. The array is sorted alphabetically.

Affected by include_hidden.

Note: When used on a `res://` path in an exported project, only the files actually included in the PCK at the given folder level are returned. In practice, this means that since imported resources are stored in a top-level `.godot/` folder, only paths to `*.gd` and `*.import` files are returned (plus a few files such as `project.godot` or `project.binary` and the project icon). In an exported project, the list of returned files will also vary depending on whether ProjectSettings.editor/export/convert_text_resources_to_binary is `true`.

PackedStringArray **get_files_at**(path: String) static 🔗

Returns a PackedStringArray containing filenames of the directory contents, excluding directories, at the given `path`. The array is sorted alphabetically.

Use get_files if you want more control of what gets included.

String **get_next()** 🔗

Returns the next element (file or directory) in the current directory.

The name of the file or directory is returned (and not its full path). Once the stream has been fully processed, the method returns an empty String and closes the stream automatically (i.e. list_dir_end would not be mandatory in such a case).

Error **get_open_error()** static 🔗

Returns the result of the last open call in the current thread.

---

int **get_space_left()** 🔗

Returns the available space on the current directory's disk, in bytes. Returns `0` if the platform-specific method to query the available space fails.

---

bool **is_case_sensitive**(path: String) const 🔗

Returns `true` if the file system or directory use case sensitive file names.

Note: This method is implemented on macOS, Linux (for EXT4 and F2FS filesystems only) and Windows. On other platforms, it always returns `true`.

bool **is_link**(path: String)

Returns `true` if the file or directory is a symbolic link, directory junction, or other reparse point.

Note: This method is implemented on macOS, Linux, and Windows.

Error **list_dir_begin**()

Initializes the stream used to list all files and directories using the get_next function, closing the currently opened stream if needed. Once the stream has been processed, it should typically be closed with list_dir_end.

Affected by include_hidden and include_navigational.

Note: The order of files and directories returned by this method is not deterministic, and can vary between operating systems. If you want a list of all files or folders sorted alphabetically, use get_files or get_directories.

void **list_dir_end**()

Closes the current stream opened with list_dir_begin (whether it has been fully processed with get_next does not matter).

Error **make_dir**(path: String)

Creates a directory. The argument can be relative to the current directory, or an absolute path. The target directory should be placed in an already existing directory (to create the full path recursively, see make_dir_recursive).